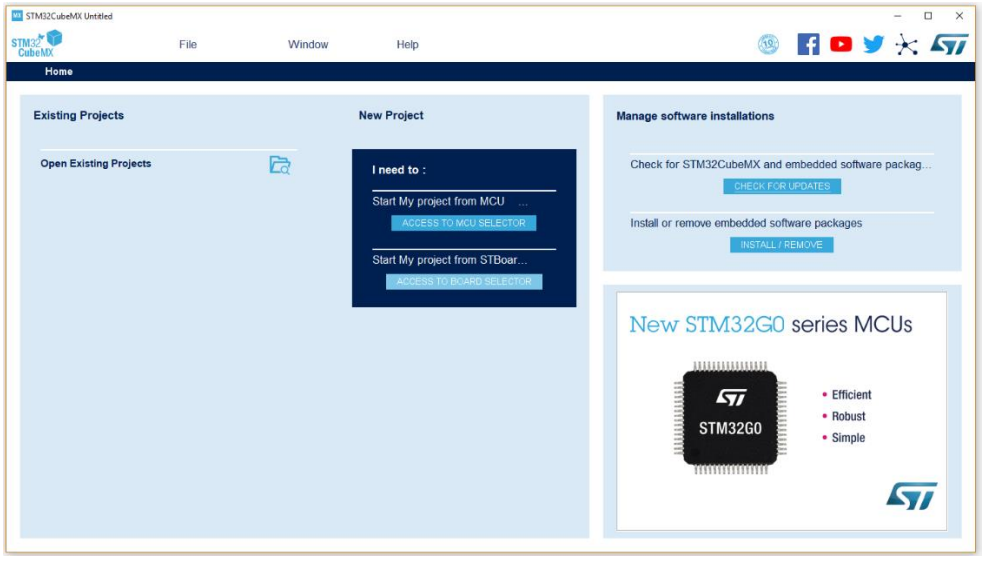
Setting up CAN and GPIO on the STM32F303

Software Versions used in this guide
STM32CubeMX: version 5.0.1
STM32Cube: version 1.0
Atollic TrueSTUDIO: version 9.2.0

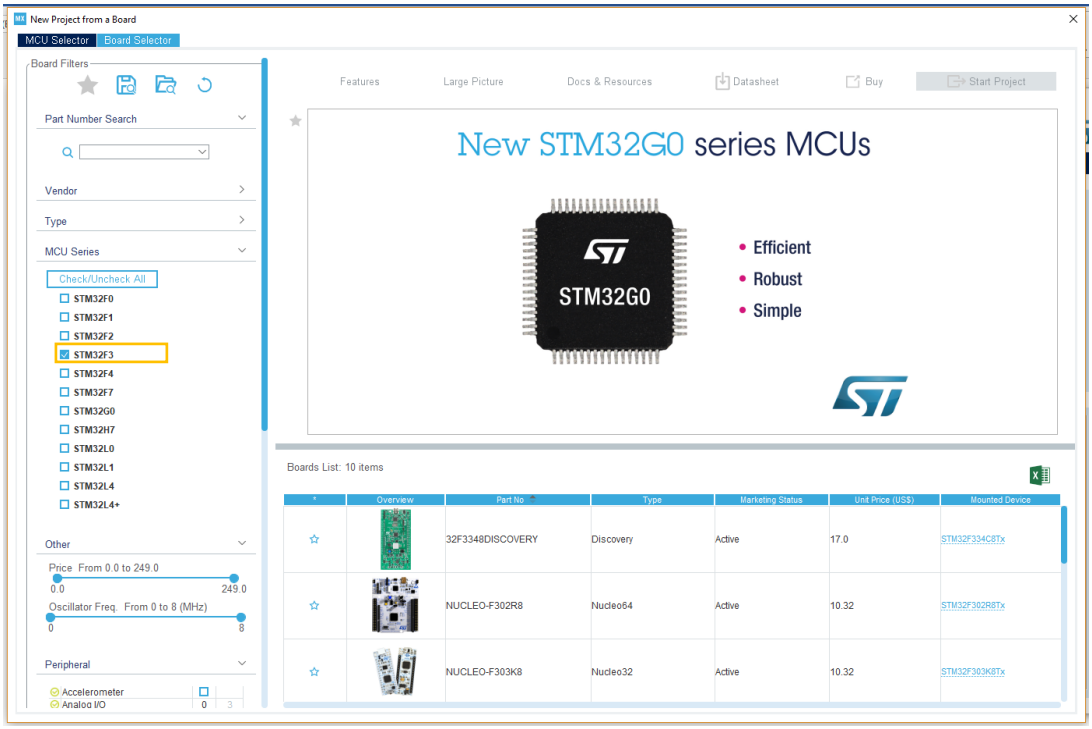
1. Install (update) Atolic TrueSTUDIO for STM32
    a. Click on Get Software and download and install: http://www.st.com/en/development-tools/truestudio.html
    b. Read the blog (during download) on the STM32Cube MCU package and STM32CubeMX at http://blog.atollic.com/best-practices-faq-truestudio-stm32cubemx

2. Install (update) STM32 CubeMx (used to easily set up code/projects for a target ARM device)
    a. https://www.st.com/en/development-tools/stm32cubemx.html
    b. Click on 'Get Software' and download STM32CubeMX – Note: you will have to sign up and get the download link in your email
    c. Unzip the downloaded file and run the installer
        i. You will need to have Java installed (install it if you don't have it)
        ii. Use all default settings

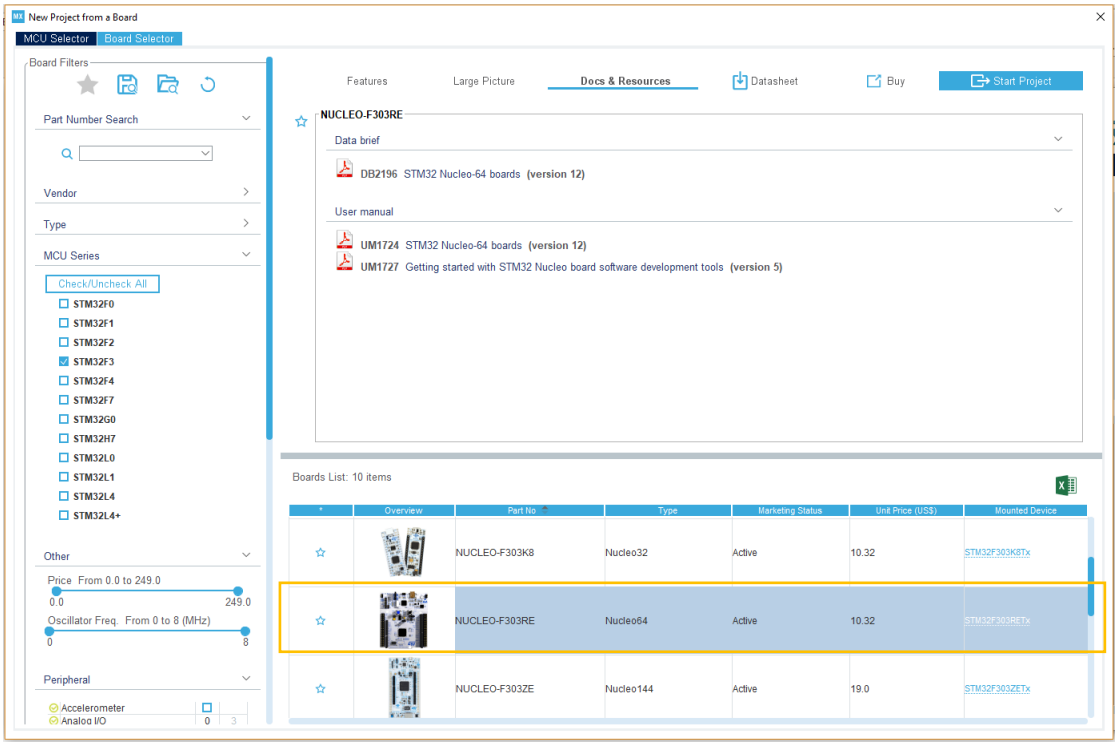
3. Open STM32 CubeMX
    a. This will be used to generate initialization code that we will open in Atollic TrueStudio, continue to modify and then deploy to the STM32 using the ST-link driver that is built into TrueStudio.
    b. After CubeMX starts, under '**New project**' click on '**Access to Board Selector**'



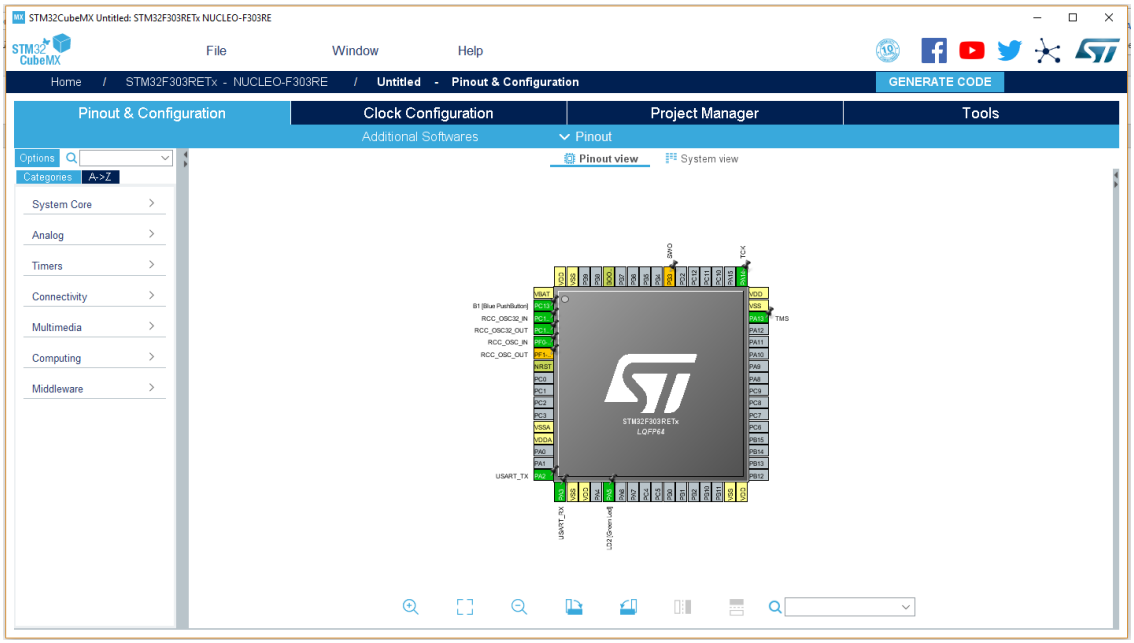
c. Under MCU Series (drop down) Click the check box for '**STM32F3**'



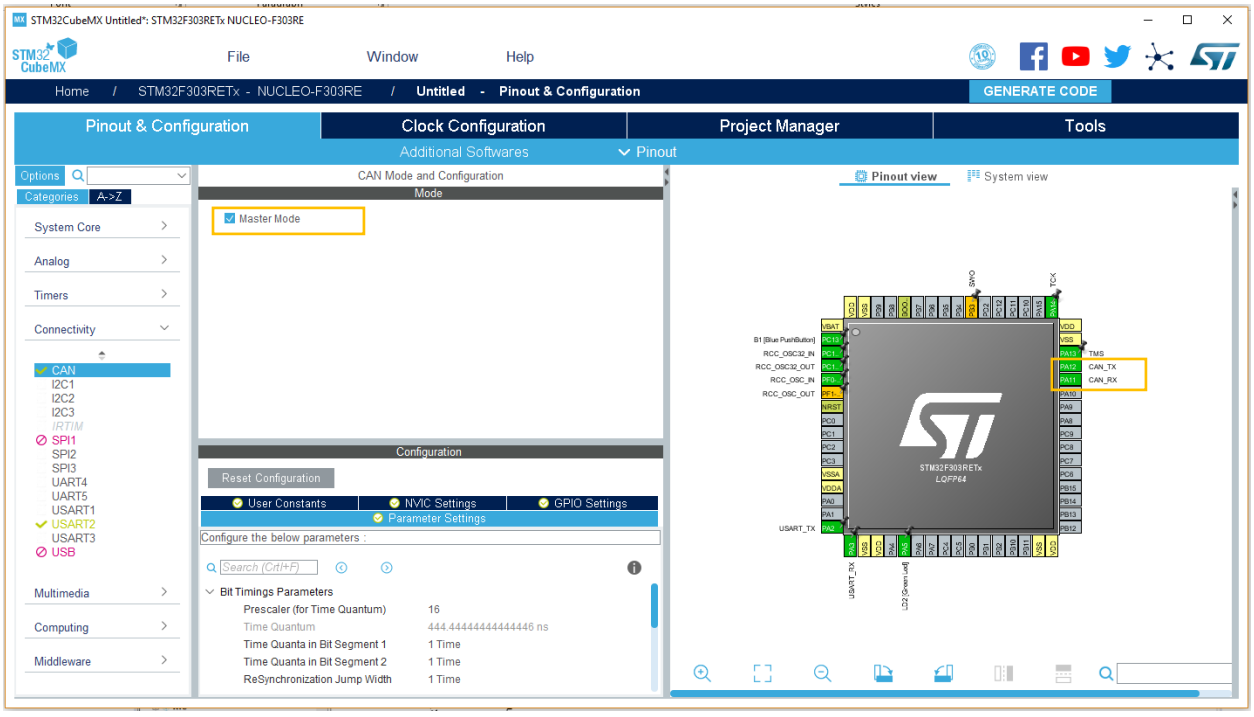
d. Scroll down in right window and select **NUCLEO-F303RE**



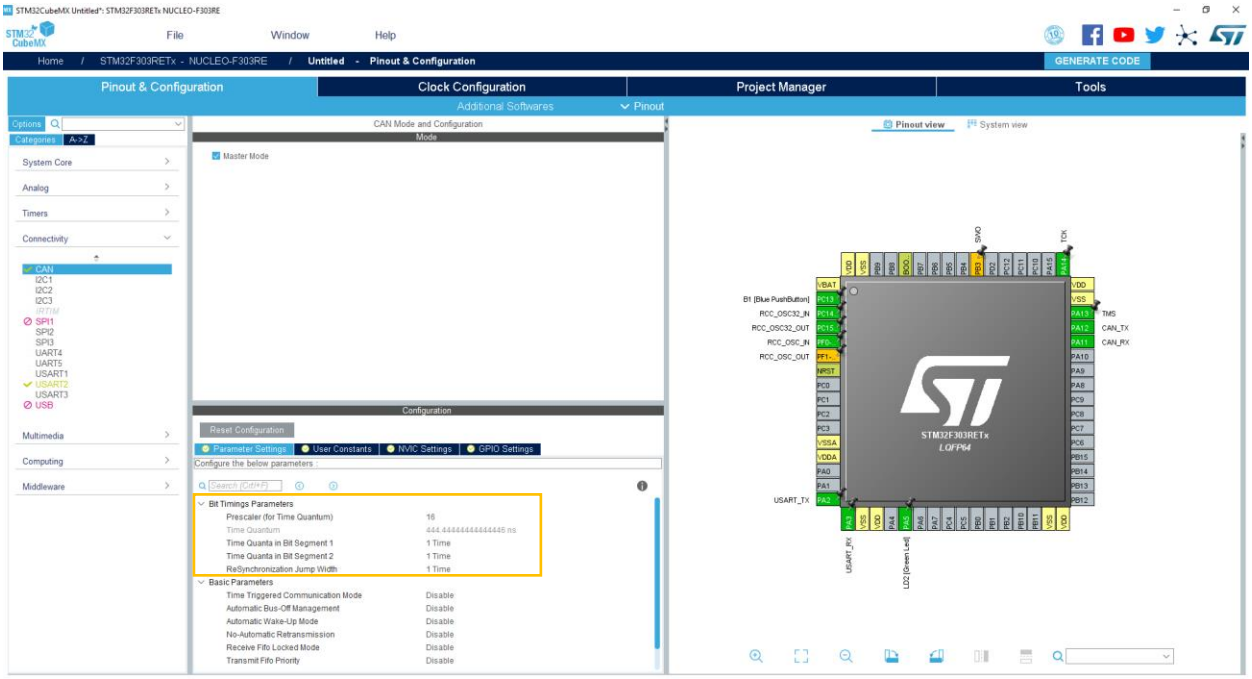
e. Download the ‘Docs and Resources’ as well as the ‘Datasheet’
f. Click on ‘Start Project’ and click on ‘Initialize peripherals to default values’
g. You will be presented with the pinout diagram as shown below



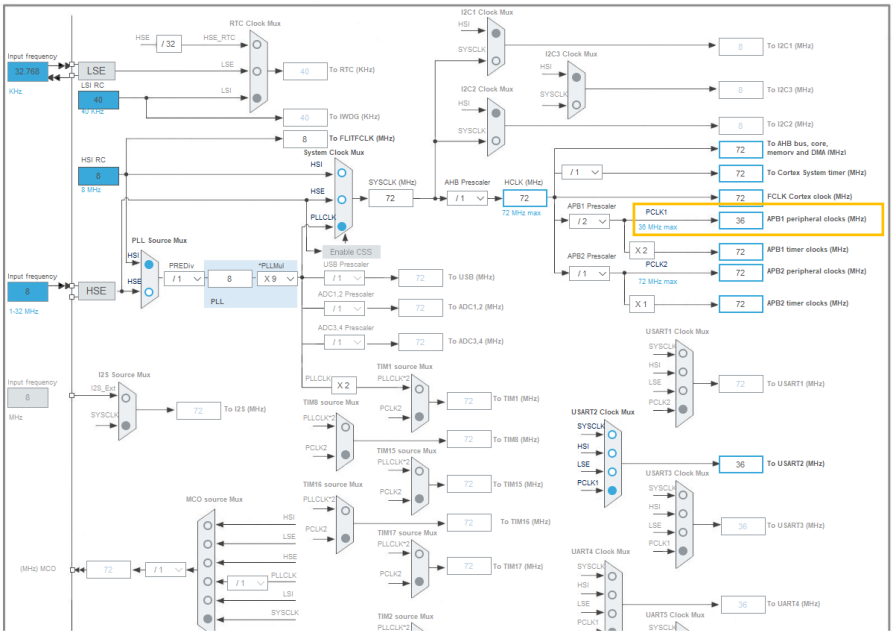
4. We will now enable CAN on PA11 and PA12
   a. Click in ‘Connectivity’ and then click on ‘CAN’
   b. Click on box that says ‘Master Node’ and PA11 and PA12 will change automatically to CAN_Tx and CAN_Rx



c. We will now configure the ‘Bit Timing Parameters’ (we will learn more about why these values are chosen later, for now we will just focus on how to set them)



d. Confirm that the CAN clock frequency (f_can) is 36 MHz by clicking on ‘Clock configuration’ and looking for output ‘APB1 Peripheral clock’. The reference manual states that the CAN bus controller clock is connected on APB1.



e. The bit timing parameters are configured according to the following values (we will study why later)

To set CAN communication bit rate (f_bit) to 125 kbps (we will see why we want this value later):

f_bit = f_can / (N * K) = 0.125 Mbps

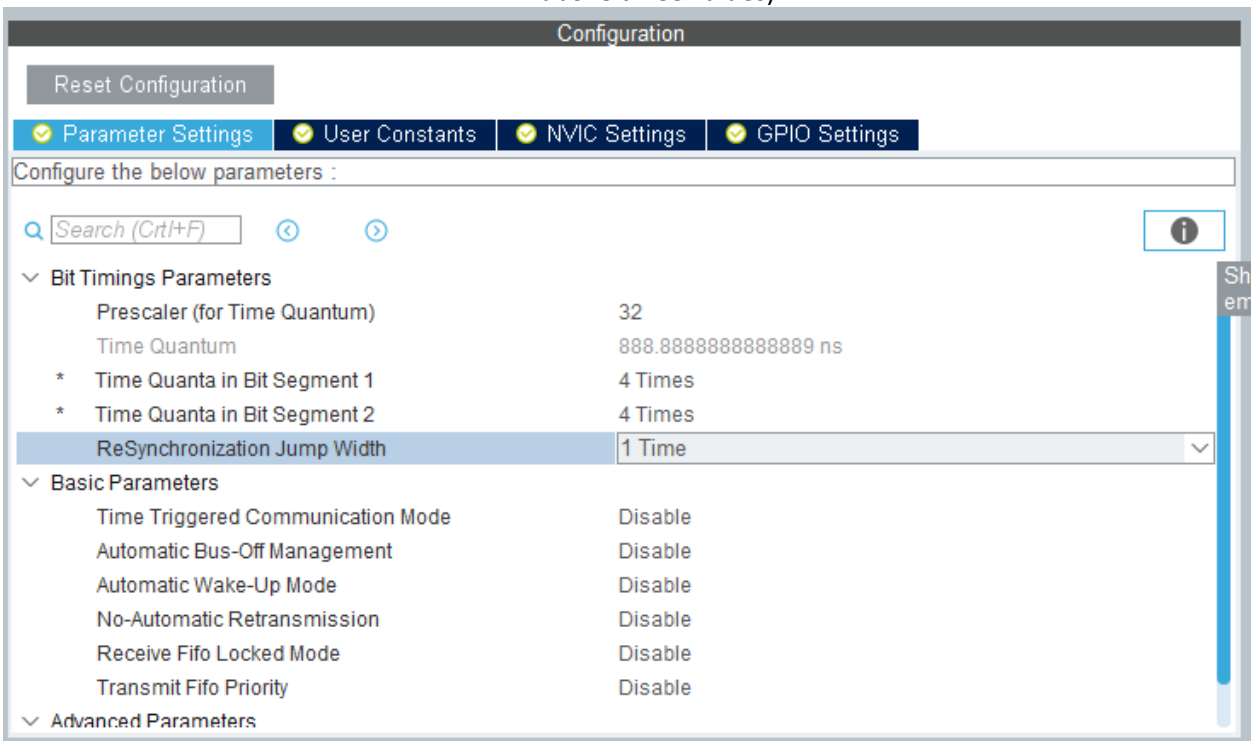
This is possible by choosing N = 9, K = 32 → f_bit = 36 MHz / (9 * 32) = 0.125

Since N=9 we need, SYNC=1, TSEG1=4, TSEG2=4 (we will see why later)

Set these values by going back to the 'Pinout & Configuration' tab and setting the values.

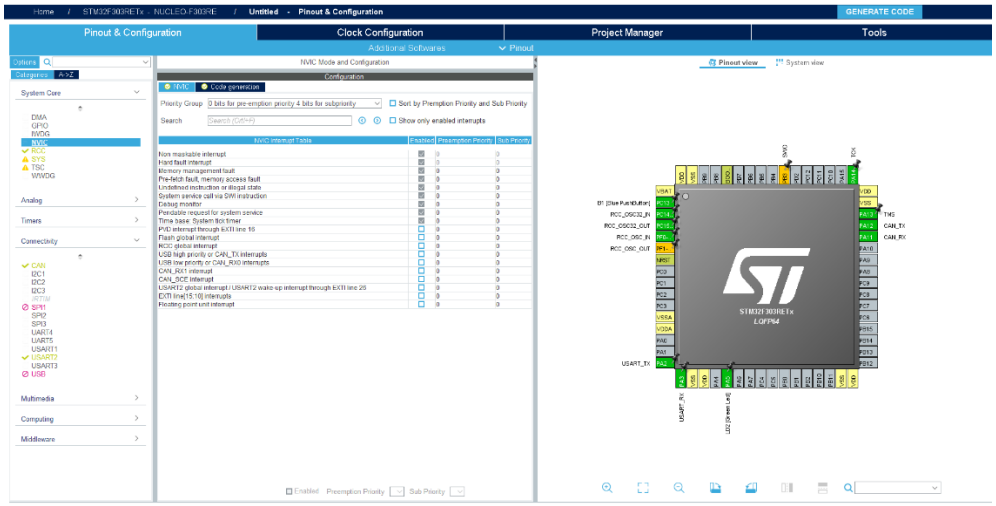
Notes:

K == '*Prescaler (for time quantum)*'
SYNC == '*ReSynchronization jump width*'
TSEG1 == '*Time quanta in bit segment 1*'
TSEG2 == '*Time quanta in bit segment 2*'
N == SYNC + TSEG1 + TSEG2 (not set directly in the configuration but is the sum of the above three values)

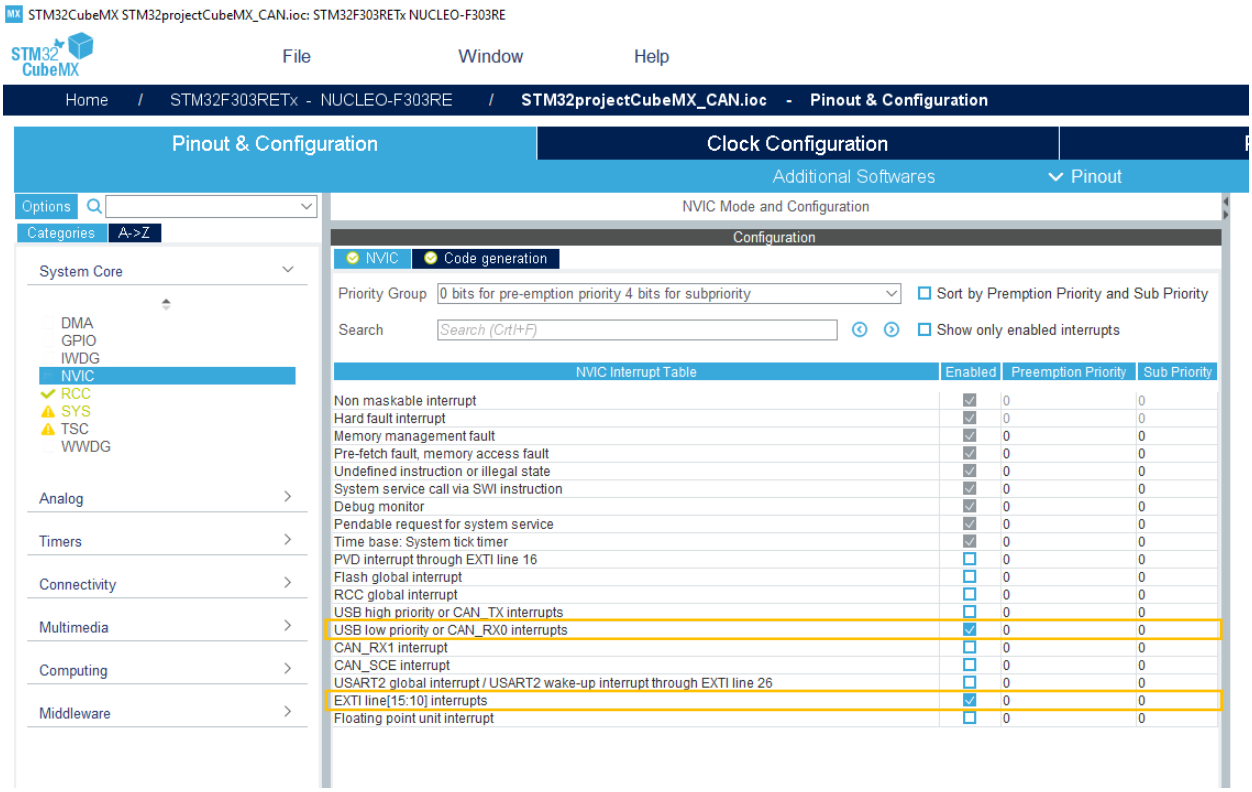


(Leave all other parameters to their defaults – *Disable* and *Normal*)

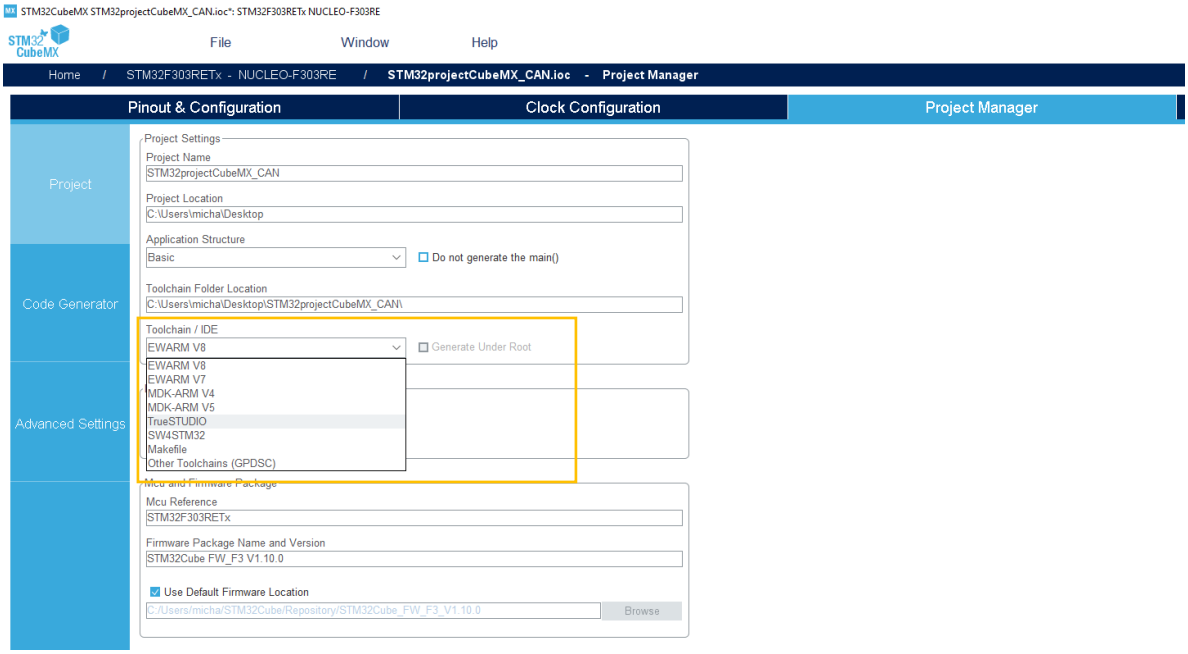
5. The next step is to set up interrupt vectors for received (Rx) messages and when a button is pressed (e.g. to transmit a message when a ‘call’ button is pressed). Click on ‘System Core’ then ‘NVIC’ - *Nested Vector Interrupt Controller*



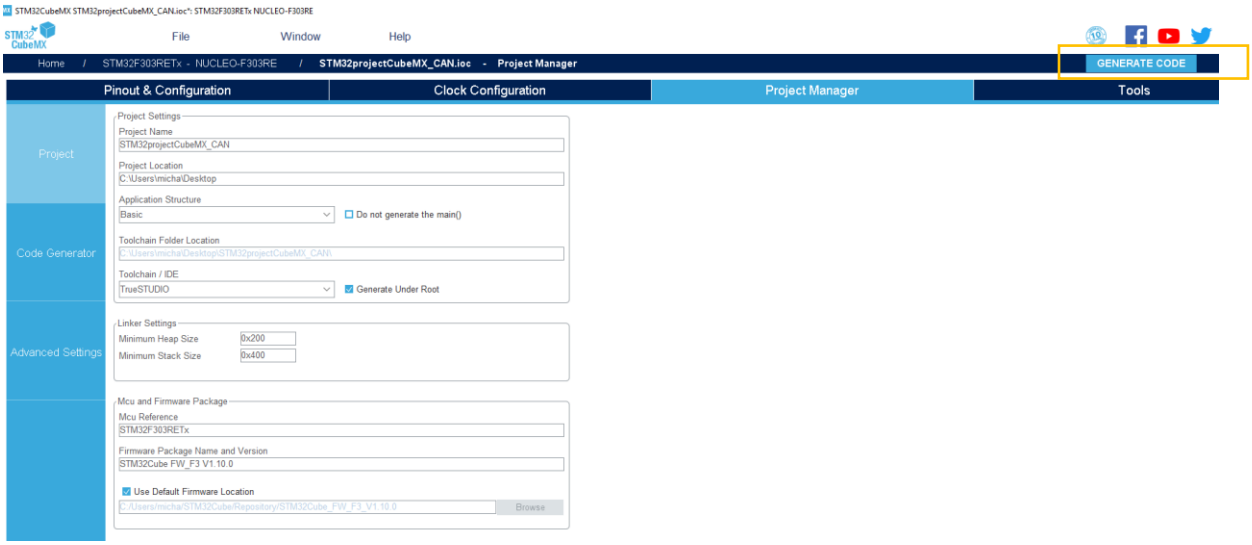
6. On the NVIC tab, click on the ‘Enable’ check box for ‘*USB low priority or CAN_RX0 interrupts*’ (USB-CAN Rx interrupts are shared between CAN and USB). This enables the Nested Vector Interrupt Controller (NVIC) for CAN Rx messages. Also on the NVIC tab, click on the ‘Enable’ check box for EXTI line[15:10] interrupts. This enables the GPIO interrupts (e.g. for the pushbutton(s)).



7. To generate the base (initialization) project code using STM32CubeMx:
    a. Click on the ‘Project Manager Tab’, then click on ‘Project’
    b. Under the Toolchain / IDE heading select ‘**TrueSTUDIO**’ (we will use this IDE to edit project files)

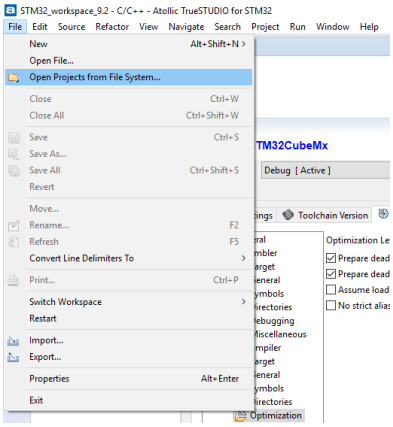


    c. Select the location and name for the deployed TrueStudio project to deploy the base initialization code (save the project).
    d. Click on Generate Code

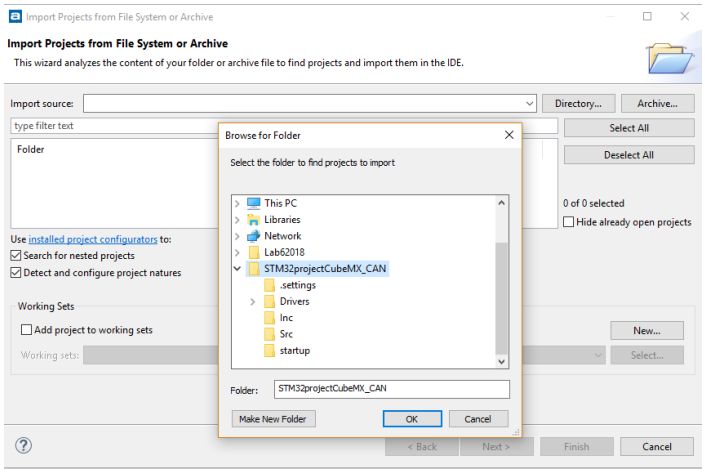


    e. Project files will be created in the saved directory (note the .ioc file is the STM32CubeMX project, you can use this later to regenerate the project and add new functionality)
    f. Close STM32CubeMx generator

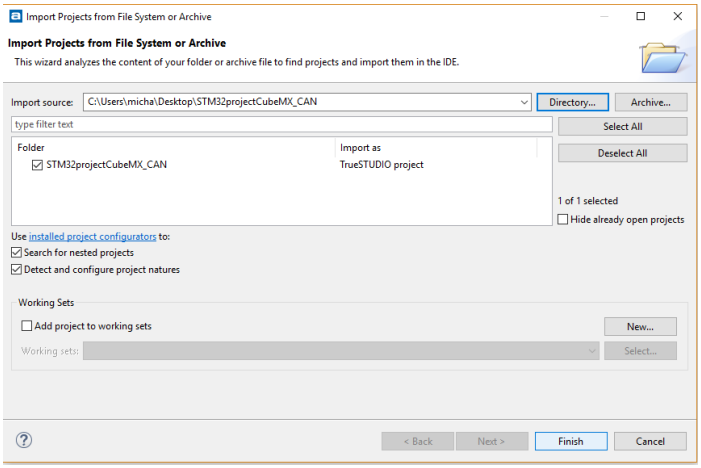
8. Open Atollic TrueStudio
    a. Open the Atollic TrueStudio project generated by STM32CubeMX by clicking on File >> Open Projects from File System



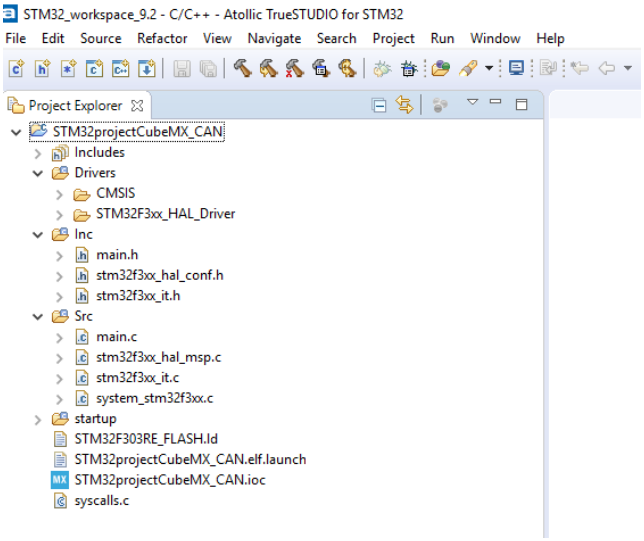
    b. Select ‘Directory’ and choose the folder you created the TrueStudio Project in



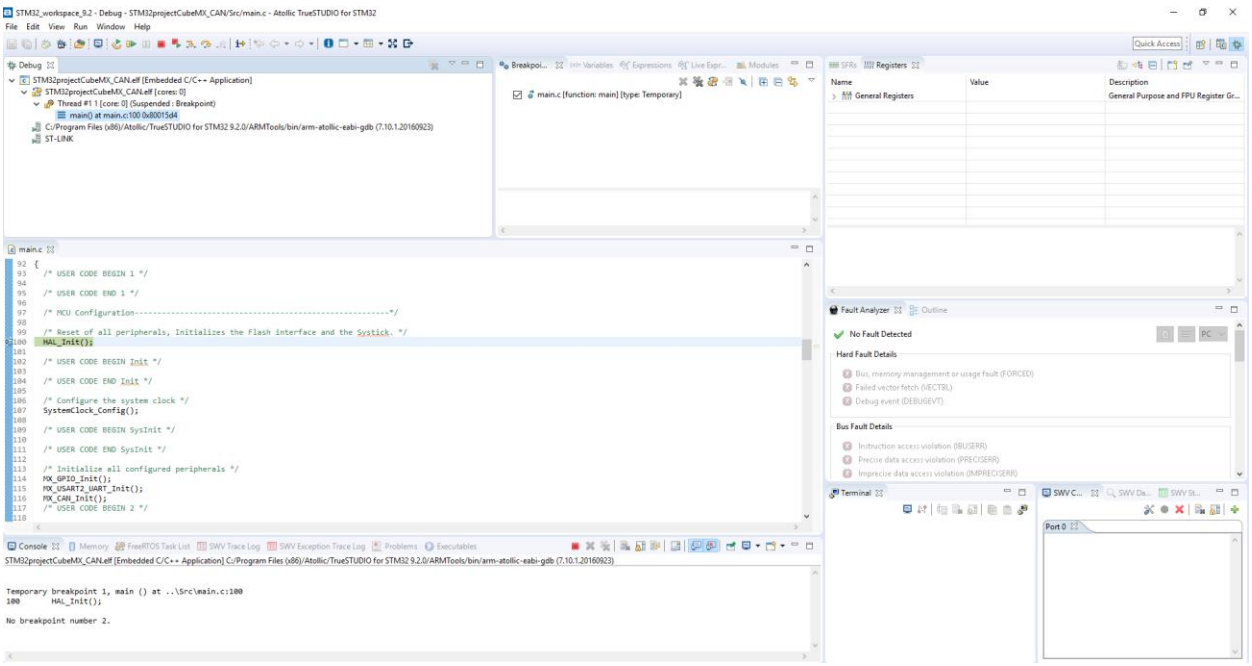
    c. Hit ‘Finish’ to open the project

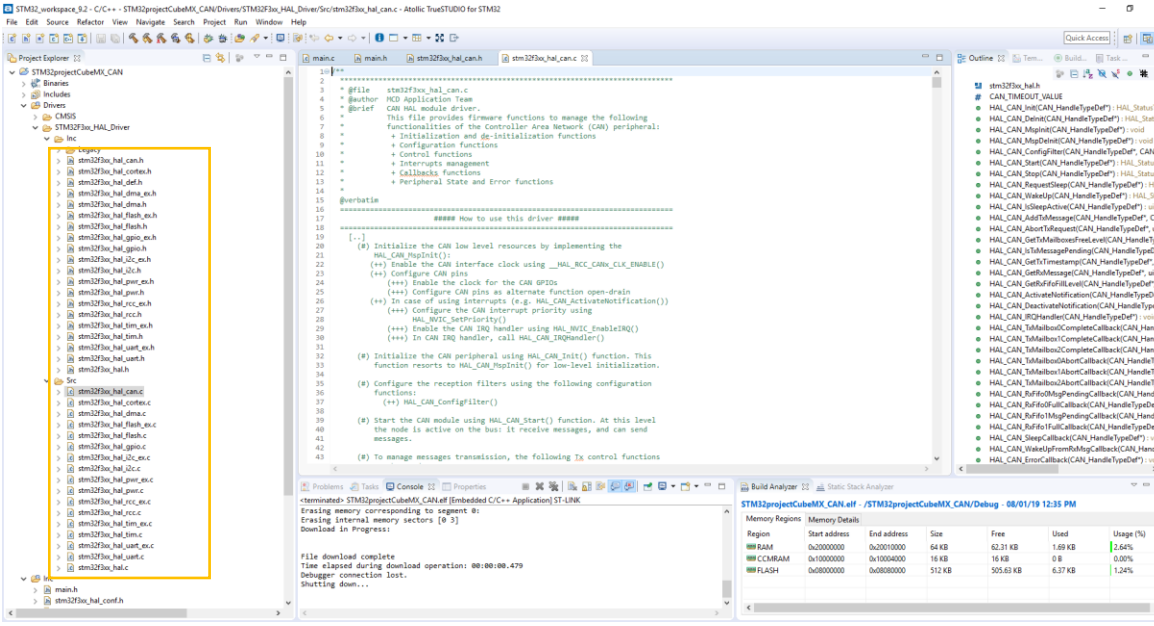


d. Close the ‘Information center’ tab to show the newly opened project
e. Project files will be loaded into Atollic TrueStudio for you to edit/add



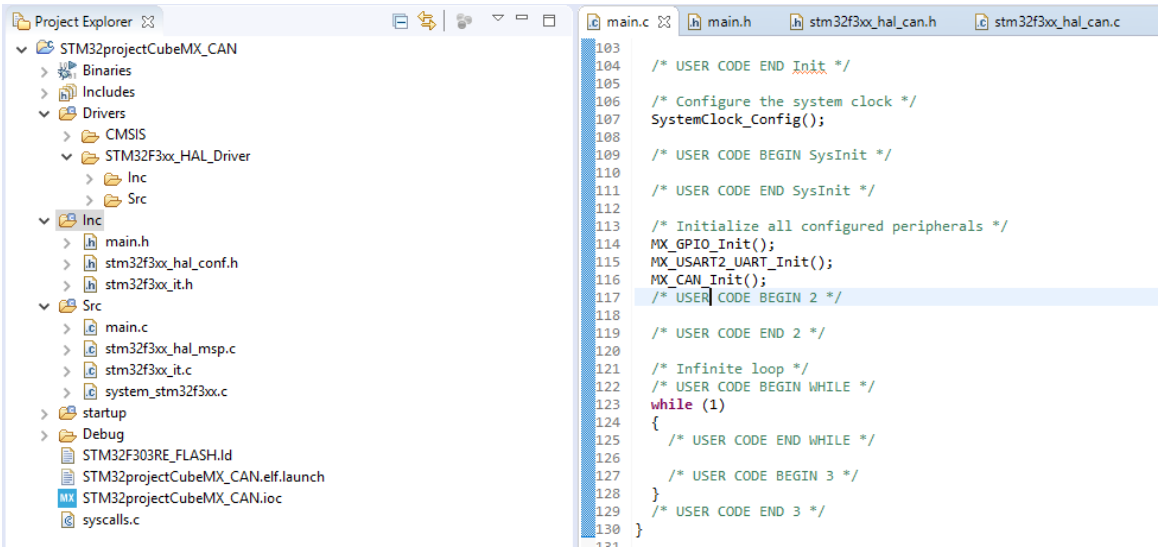
9. Test the default code to make sure it can be deployed to the STM32
    a. Connect to the STM32 using the USB cable
    b. Note: You may need to update ST-Link within Atollic TrueStudio and/or the Firmware on your STM32F303RE Board (it will ask you and update automatically, if required)
    c. Click ‘Project >> Build Project’ (this compiles the code)
    d. Select ‘Run >> Debug as >> Embedded C/C++ application’ (this transfers the program to the STM32 and opens the debugger)
    e. If successful you will see the code running and the fault analyser will show ‘No fault detected’ as shown below



10. Note: The HAL driver functions/headers are located under ‘Drivers >> STM32Fxx_HAL_Driver >> Src / Inc’
    i. The main.c program includes stm32f3xx_hal.h which STM32CubeMx edits to include any other required functionality (i.e. includes stm32f3xx_hal_can.h for CAN functionality and stm32f3xx_hal_gpio.h for GPIO (LED, Pushbutton) functionality).
    ii. The notes at the top of each driver file (e.g ‘stm32f3xx_hal_can.c’) describe usage of the functions. More detailed usage notes are in the PDFs. We will learn a few basic functions for CAN and GPIO functionality in the next steps.
    iii. 

11. Now we will start adding the CAN Tx/Rx functionality. The main.c file is in the top level ‘Src’ directory and the header files are in the top level ‘Inc’ directory. Note: YOU MUST ONLY ADD YOUR USER CODE BETWEEN THE COMMENTS. Any code outside these comments will be OVERWRITTEN by STM32CubeMX if you update the initialization code.

```
/* USER CODE BEGIN */

        Your code goes here …

/* USER CODE END */
```



12. The first code we will add to our project is to add defines for our communication protocol between USER CODE private define comments. The value of the ID will need to be changed later according to the communication protocol, depending on the floor number the STM32 board is assigned to. The BUTTON defines are the possible values for a flag used to indicate the press of a button.

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define     ID                  0x0100          // ID of supervisory controller - Change this depending on floor #
#define     GO_TO_FLOOR_1       0x05            // Floor 1
#define     GO_TO_FLOOR_2       0x06            // Floor 2
#define     GO_TO_FLOOR_3       0x07            // Floor 3
#define     NO_BUTTON_PRESSED       0           // Default value of the BUTTON flag - no button pressed
#define     BLUE_BUTTON_PRESSED     1           // Value of BUTTON when the blue button is pressed (add other buttons)
```

13. Next, we will create some private variables between the USER CODE private variable comments

```
/* USER CODE BEGIN PV */
CAN_TxHeaderTypeDef   TxHeader;
CAN_RxHeaderTypeDef   RxHeader;
uint8_t               TxData[8];
uint8_t               RxData[8];
uint32_t              TxMailbox;

uint8_t msg = GO_TO_FLOOR_1;                // Message is 'Go to Floor 1'
uint8_t BUTTON = NO_BUTTON_PRESSED;         // Button pressed flag (value changed in callback function for given interrupt)

/* USER CODE END PV */
```

CAN_TxHeaderTypeDef and **CAN_TxHeaderTypeDef** are data types (structs) defined in stm32f3xx_hal_can.h. They contain the header information for CAN Tx/Rx messages. TxHeader and RxHeader are variables of this type. The 8 bit CAN messages are stored in the TxData[] and RxData[] buffers (we will only use the first element of these arrays (i.e. TxData[0] and RxData[0]) in this exercise but all 8 elements can contain information).

The msg variable is the message to be transmitted from the TxData[0] buffer to the RxData[0] buffer over the CAN bus.

The BUTTON variable is a flag that will be changed in an interrupt callback function (i.e. when a button is pressed) and the desired action will be performed in main().

14. Next, we will add functionality between the /* USER CODE BEGIN 3 */ and /* USER CODE END 3 */ comments, which are inside the infinite while loop.

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    uint8_t i;
    // Receive
    if (RxData[0] == GO_TO_FLOOR_1) {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);                                  // Turn on LED2
        HAL_Delay(2000);                                                        // Keep LED on for 2 seconds
        for (i=0; i<8; i++) {
          RxData[i] = 0x00;                                                     // Reset the RxData[] buffer (used as flag)
        }
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);                                  // Turn off LED2
        HAL_Delay(100);                                                         // Need a delay after toggle
     }

     // Transmit
     if (BUTTON != 0) {
         if (BUTTON == BLUE_BUTTON_PRESSED) {                                   // Blue button pressed --> Turn on LED2 for 2 seconds and Transmit message
             HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);                             // Turn on LED2
             HAL_Delay(2000);                                                   // Leave it on for 2 seconds
             TxData[0] = msg;                                                   // Store the 1 character message to transmit into the TxData buffer and transmit over the CAN b
             if (HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox) != HAL_OK) {    // Transmit the message
                Error_Handler();                                                // Transmission error
             }
             HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);                             // Turn off LED2
             BUTTON = NO_BUTTON_PRESSED;                                        // Reset the BUTTON flag
         }
     }
  }
  /* USER CODE END 3 */
}
```

Receive

We have not yet written the callback function for the receive interrupt, however, this function will store the value of the received message in RxData[0]. So when RxData[0] changes from 0 (default for 'No data') to GO_TO_FLOOR_1 (0x05), we will indicate that the message has been received by turning on LED2 for 2 seconds and then clearing the RxData[] buffer.

Transmit

We have not yet written the callback function that is called by the interrupt that occurs when the blue button is pressed. The callback function will change the value of the BUTTON variable from NO_BUTTON_PRESSED (0) to another value (e.g. BLUE_BUTTON_PRESSED (1)).

15. We will now add user code to the MX_CAN_INIT() function generated by STM32CubeMx. Add the following code between the /* USER CODE BEGIN CAN_Init 2 */ comments.

```
/* USER CODE BEGIN CAN_Init 2 */
/******************************************************************************/

/* *** Set up CAN Rx filters *** */
CAN_FilterTypeDef filter;                       // This is one of the 13 filters - can create more filters - this one will be number 0

/* Configure filter 0 to direct everything to FIFO 0 */
filter.FilterBank = 0;                          // This is filter number 0
filter.FilterIdHigh = 0x0000;                   // All bits 0 - dont care about any - See page 1041 in reference manual for how to set these bits & mask bits in the register
filter.FilterIdLow = 0x0000;
filter.FilterMaskIdHigh = 0x0000;
filter.FilterMaskIdLow = 0x0000;
filter.FilterFIFOAssignment = CAN_FILTER_FIFO0;
filter.FilterMode = CAN_FILTERMODE_IDMASK;      // uses mask mode (so can set range of IDs)
filter.FilterScale = CAN_FILTERSCALE_32BIT;     // Use 32 bit filters
filter.FilterActivation = ENABLE;               // By default the filters are disabled so enable them
filter.SlaveStartFilterBank = 0;

if(HAL_CAN_ConfigFilter(&hcan, &filter) != HAL_OK) {  // Set the above values for filter 0
  Error_Handler();
}

/* *** Start the CAN peripheral *** */
if (HAL_CAN_Start(&hcan) != HAL_OK) {
    Error_Handler();
}

/* *** Activate CAN Rx notification interrupt *** */
if (HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO0_MSG_PENDING) != HAL_OK) {
    Error_Handler();
}

/* *** Prepare header fields for Standard Mode CAN Transmission *** */
TxHeader.IDE = CAN_ID_STD;                      // Using standard mode. Note this = CAN_ID_EXT for extended mode
TxHeader.ExtId = 0x00;                          // Extended ID is not used
TxHeader.StdId = ID;                            // Standard mode ID is 0x100 -- CHANGE THIS LATER ---
TxHeader.RTR = CAN_RTR_DATA;                    // Send a data frame not an RTR
TxHeader.DLC = 1;                               // Data length code = 1 (only send one byte)
TxHeader.TransmitGlobalTime = DISABLE;

/******************************************************************************/
/* USER CODE END CAN_Init 2 */
```

The CAN_FilterTypeDef data type (struct) is defined in **stm32f3xx_hal_can.h**. This struct is used to define filters used to accept / reject CAN messages. The function HAL_CAN_ConfigFilter() is used to set the values in the struct for the specified filter (filter number 0). HAL_CAN_Start() starts the CAN peripheral. HAL_CAN_ActivateNotification() is used to activate the CAN Rx interrupt in FIFO0 (interrupt vector is CAN_IT_RX_FIFO0_MSG_PENDING). The header fields for the message to be transmitted are also set in TxHeader.

16. In STM32CubeMx we enabled the '*USB low priority or CAN RX0 interrupts*'. When a CAN Rx interrupt occurs it calls the function USB_LP_CAN_RX0_IRQHandler() in stm32f3xx_it.c, which in turn calls HAL_CAN_IRQHandler() in stm32f3xx_hal_can.c, which clears the interrupt flag and calls the callback function HAL_CAN_RxFifo0MsgPendingCallback(). A callback function is one that must be overwritten in order to do something when an interrupt is triggered. We will now override the CAN Rx callback function HAL_CAN_RxFifo0MsgPendingCallback(). Add the following lines below /* USER CODE BEGIN 4 */

```
/* USER CODE BEGIN 4 */

/******************************************************************************************** */

// Override the HAL_CAN_RxFifo0MsgPendingCallback function.
// This is called when the interrupt for FIFO0 is triggered.
/******************************************************************************************** */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    /* Get RX message and store in RxData[] buffer */
      if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, RxData) != HAL_OK)
      {
        /* Reception Error */
        Error_Handler();
      }
}
```

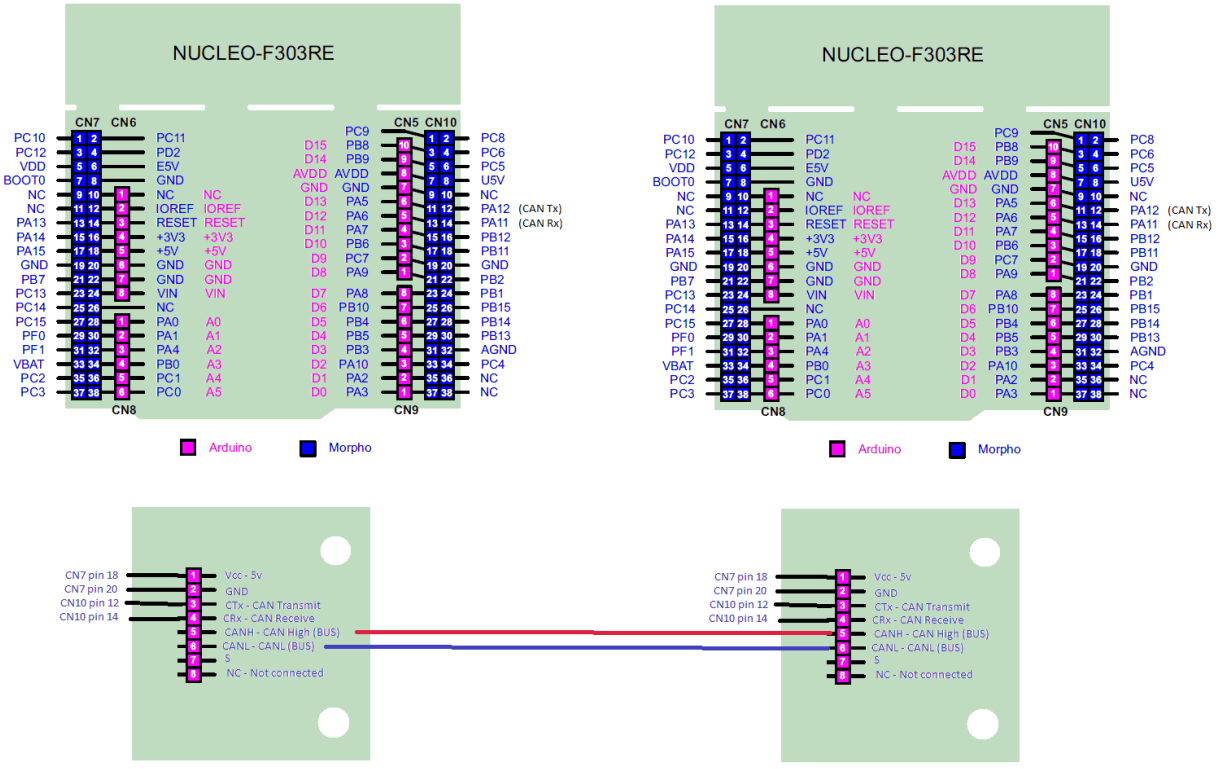
The callback function calls HAL_CAN_GetRxMessage() to store the received message in RxData[].

17. In STM32CubeMx we enabled the *'EXTI Line [15:10] interrupts'*. These are used for GPIO interrupts (e.g. button press). When an EXTI_15_10 interrupt occurs, it calls the function EXTI15_10_IRQHandler() in stm32f3xx_it.c, which in turn calls HAL_GPIO_EXTI_IRQHandler() in stm32f3xx_hal_gpio.c, which clears the interrupt flag and calls the callback function HAL_GPIO_EXTI_Callback(). This function must be overwritten in order to do something when an interrupt is triggered. Add the following lines of code below the code above but before /* USER CODE END 4 */

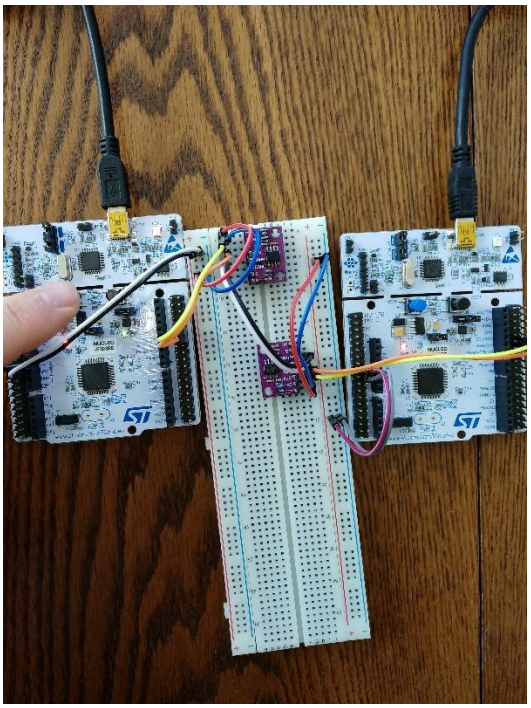
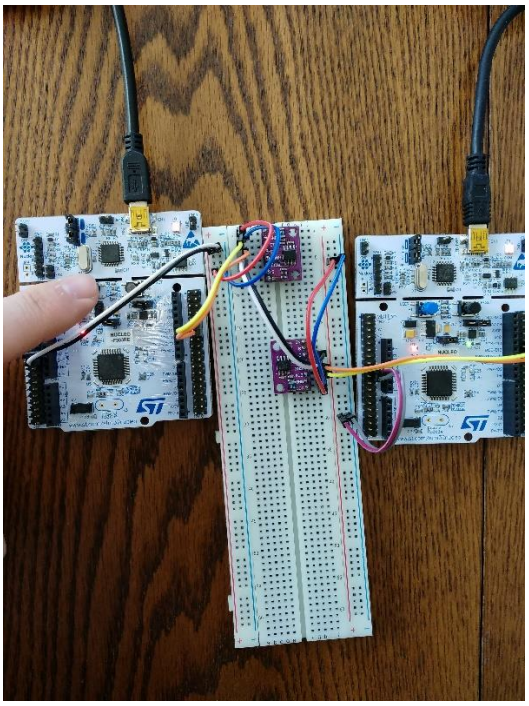
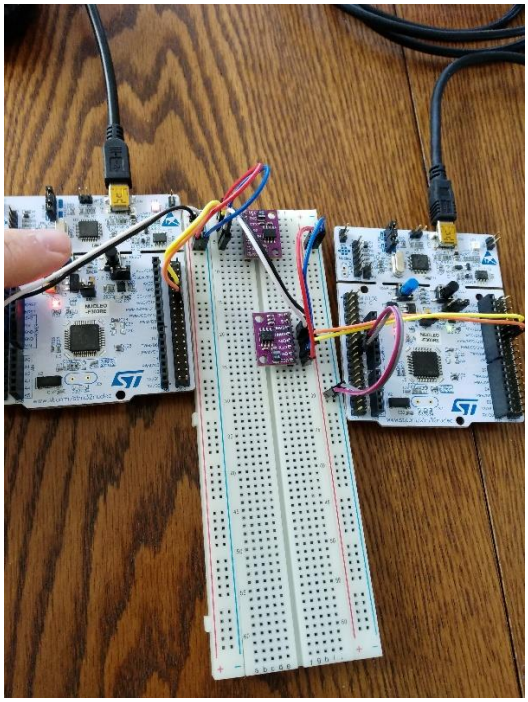
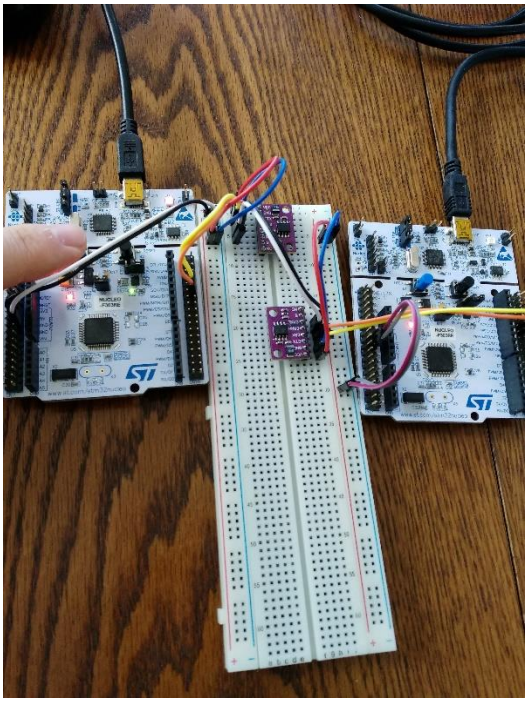
```
// Override the HAL_GPIO Callback
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  // Set the BUTTON Flag to indicate which button was pressed
  if (GPIO_Pin == GPIO_PIN_13)                  // GPIO pin 13 is the blue push button
  {
      BUTTON = BLUE_BUTTON_PRESSED;                              // Blue button pressed
  }

}
```

When the BUTTON flag is changed to BLUE_BUTTON_PRESSED the required action (turn on LED and transmit message) it is handled in main().

18. Build and run the project and deploy it to two STM32F303RE boards. When you press the blue button, the green LED2 should light up for 2 seconds.
19. Next we will connect CAN transceivers to the STM32 boards and connect them together using the diagram below.



20. When everything is connected, pressing the blue button on one STM will light up LED2 on both STMs for 2 seconds (two seconds to for Tx side and another 2 seconds on Rx side).

References

STM32F303RE Page: https://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32-mainstream-mcus/stm32f3-series/stm32f303/stm32f303re.html#sw-tools-scroll