

## Setting up CAN and GPIO on the STM32F303

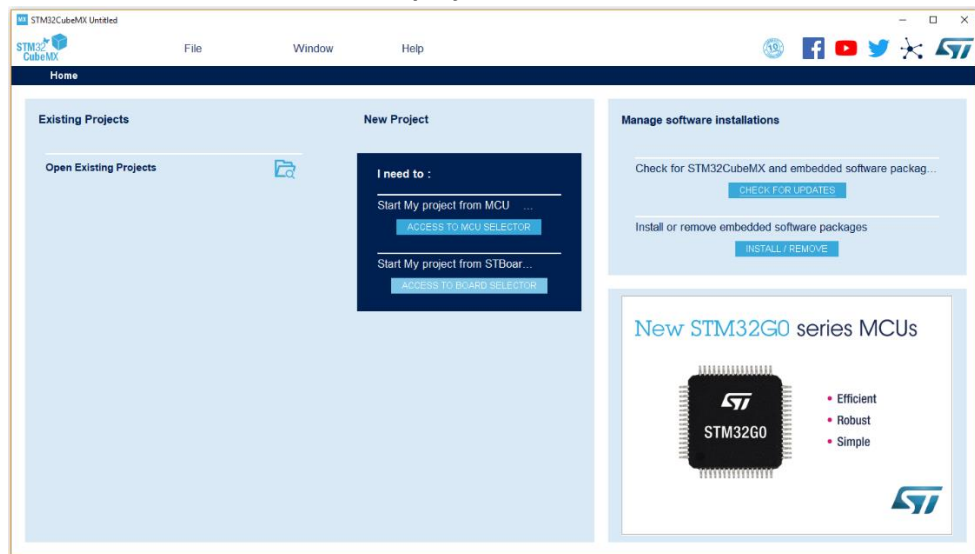
Software Versions used in this guide

STM32CubeMX: version 5.0.1

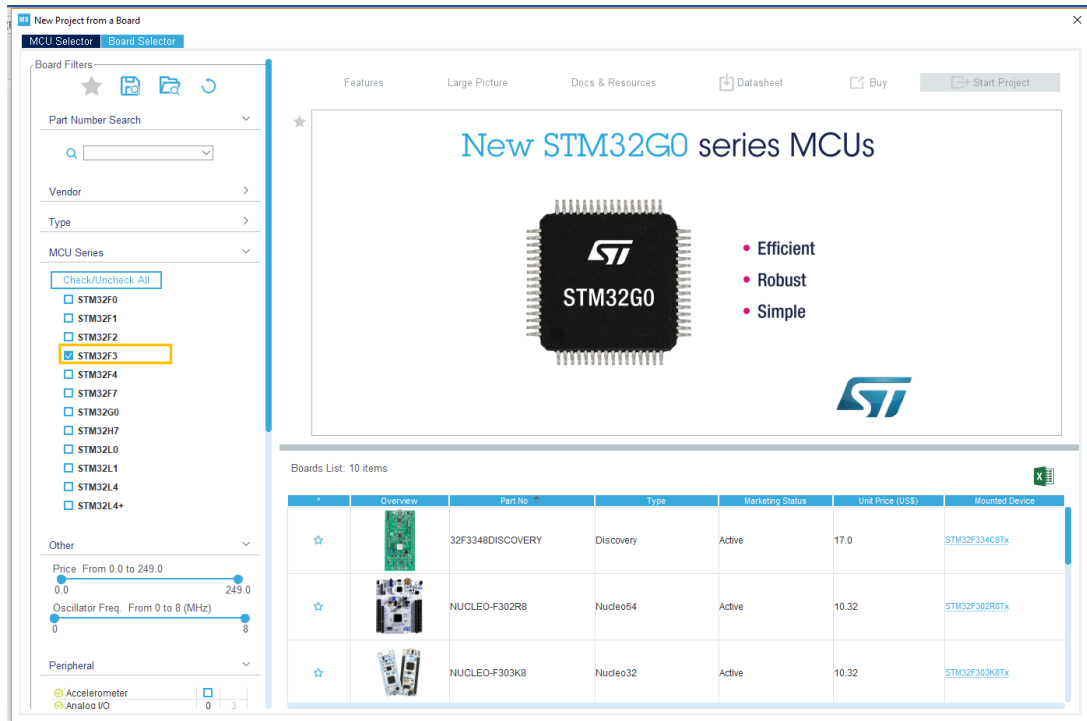
STM32Cube: version 1.0

Atollic TrueSTUDIO: version 9.2.0

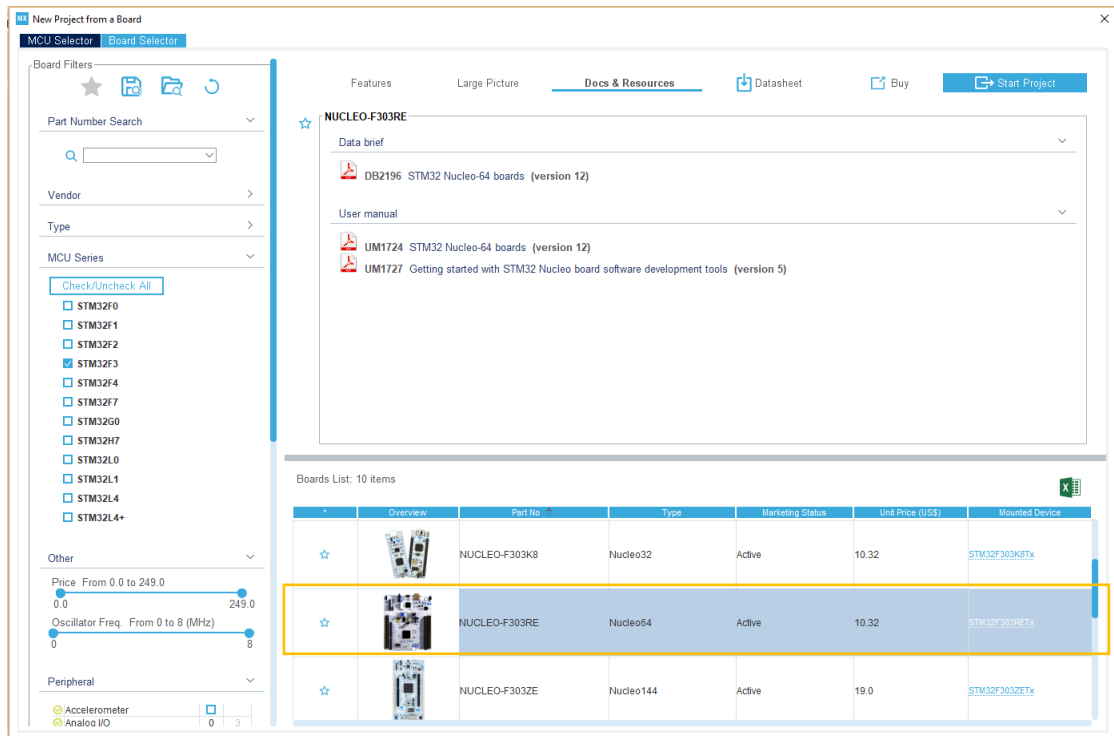
1. Install (update) Atollic TrueSTUDIO for STM32
  - a. Click on Get Software and download and install: <http://www.st.com/en/development-tools/truestudio.html>
  - b. Read the blog (during download) on the STM32Cube MCU package and STM32CubeMX at <http://blog.atollic.com/best-practices-faq-truestudio-stm32cubemx>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



- STM32CubeMX

File

Window

Help

Home / STM32F303RETx - NUCLEO-F303RE / Untitled - Pinout & Configuration

GENERATE CODE

Pinout & Configuration

Clock Configuration

Project Manager

Tools

Options

Categories

System Core

Analog

Timers

Connectivity

Multimedia

Computing

Middleware

Additional Softwares

Pinout

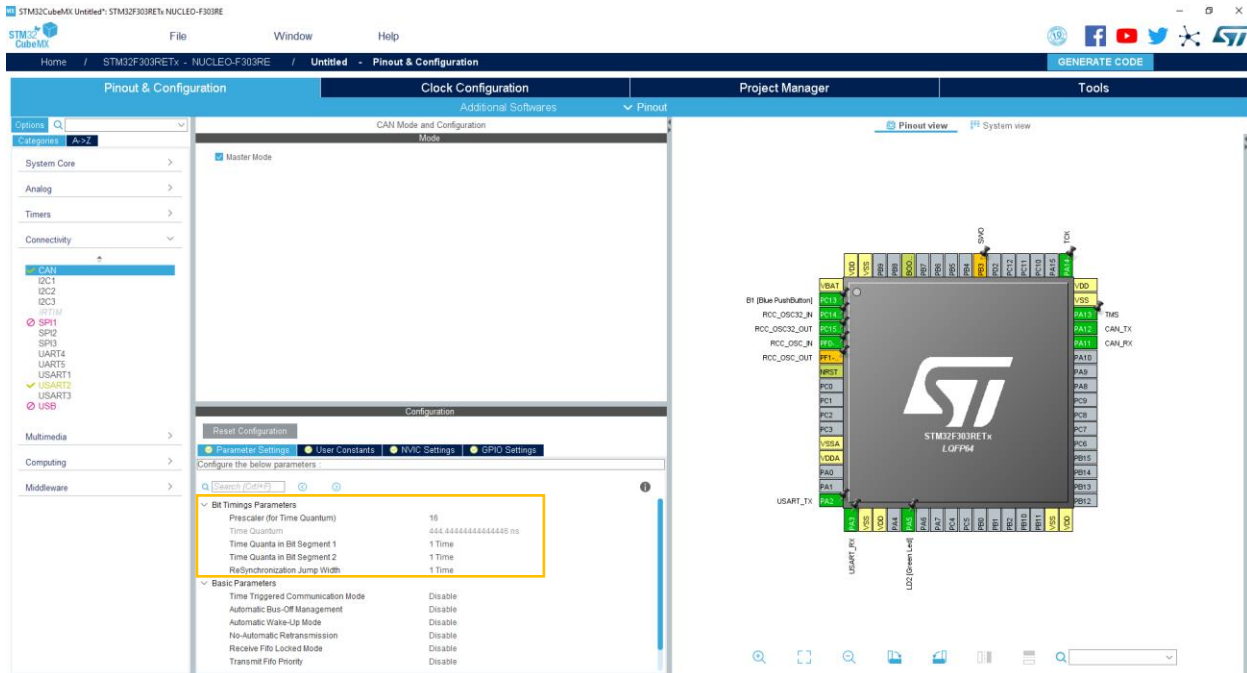
Pinout view

System view

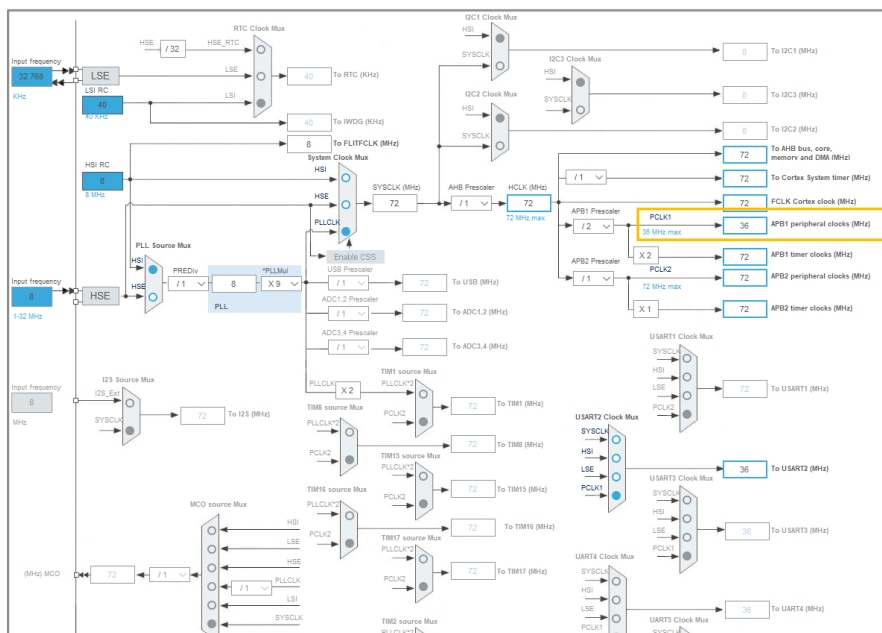
Pinout diagram of the STM32F303RETx NUCLEO-F303RE. The diagram shows the microcontroller chip with its pins and their functions. The pins are color-coded: blue for UART, green for I2C, yellow for SPI, and red for other functions. The functions listed on the left include: UART1, UART2, UART3, UART4, UART5, UART6, UART7, UART8, UART9, UART10, UART11, UART12, UART13, UART14, UART15, UART16, UART17, UART18, UART19, UART20, UART21, UART22, UART23, UART24, UART25, UART26, UART27, UART28, UART29, UART30, UART31, UART32, UART33, UART34, UART35, UART36, UART37, UART38, UART39, UART40, UART41, UART42, UART43, UART44, UART45, UART46, UART47, UART48, UART49, UART50, UART51, UART52, UART53, UART54, UART55, UART56, UART57, UART58, UART59, UART60, UART61, UART62, UART63, UART64, UART65, UART66, UART67, UART68, UART69, UART70, UART71, UART72, UART73, UART74, UART75, UART76, UART77, UART78, UART79, UART80, UART81, UART82, UART83, UART84, UART85, UART86, UART87, UART88, UART89, UART90, UART91, UART92, UART93, UART94, UART95, UART96, UART97, UART98, UART99, UART100, UART101, UART102, UART103, UART104, UART105, UART106, UART107, UART108, UART109, UART110, UART111, UART112, UART113, UART114, UART115, UART116, UART117, UART118, UART119, UART120, UART121, UART122, UART123, UART124, UART125, UART126, UART127, UART128, UART129, UART130, UART131, UART132, UART133, UART134, UART135, UART136, UART137, UART138, UART139, UART140, UART141, UART142, UART143, UART144, UART145, UART146, UART147, UART148, UART149, UART150, UART151, UART152, UART153, UART154, UART155, UART156, UART157, UART158, UART159, UART160, UART161, UART162, UART163, UART164, UART165, UART166, UART167, UART168, UART169, UART170, UART171, UART172, UART173, UART174, UART175, UART176, UART177, UART178, UART179, UART180, UART181, UART182, UART183, UART184, UART185, UART186, UART187, UART188, UART189, UART190, UART191, UART192, UART193, UART194, UART195, UART196, UART197, UART198, UART199, UART200, UART201, UART202, UART203, UART204, UART205, UART206, UART207, UART208, UART209, UART210, UART211, UART212, UART213, UART214, UART215, UART216, UART217, UART218, UART219, UART220, UART221, UART222, UART223, UART224, UART225, UART226, UART227, UART228, UART229, UART230, UART231, UART232, UART233, UART234, UART235, UART236, UART237, UART238, UART239, UART240, UART241, UART242, UART243, UART244, UART245, UART246, UART247, UART248, UART249, UART250, UART251, UART252, UART253, UART254, UART255, UART256, UART257, UART258, UART259, UART260, UART261, UART262, UART263, UART264, UART265, UART266, UART267, UART268, UART269, UART270, UART271, UART272, UART273, UART274, UART275, UART276, UART277, UART278, UART279, UART280, UART281, UART282, UART283, UART284, UART285, UART286, UART287, UART288, UART289, UART290, UART291, UART292, UART293, UART294, UART295, UART296, UART297, UART298, UART299, UART300, UART301, UART302, UART303, UART304, UART305, UART306, UART307, UART308, UART309, UART310, UART311, UART312, UART313, UART314, UART315, UART316, UART317, UART318, UART319, UART320, UART321, UART322, UART323, UART324, UART325, UART326, UART327, UART328, UART329, UART330, UART331, UART332, UART333, UART334, UART335, UART336, UART337, UART338, UART339, UART340, UART341, UART342, UART343, UART344, UART345, UART346, UART347, UART348, UART349, UART350, UART351, UART352, UART353, UART354, UART355, UART356, UART357, UART358, UART359, UART360, UART361, UART362, UART363, UART364, UART365, UART366, UART367, UART368, UART369, UART370, UART371, UART372, UART373, UART374, UART375, UART376, UART377, UART378, UART379, UART380, UART381, UART382, UART383, UART384, UART385, UART386, UART387, UART388, UART389, UART390, UART391, UART392, UART393, UART394, UART395, UART396, UART397, UART398, UART399, UART400, UART401, UART402, UART403, UART404, UART405, UART406, UART407, UART408, UART409, UART410, UART411, UART412, UART413, UART414, UART415, UART416, UART417, UART418, UART419, UART420, UART421, UART422, UART423, UART424, UART425, UART426, UART427, UART428, UART429, UART430, UART431, UART432, UART433, UART434, UART435, UART436, UART437, UART438, UART439, UART440, UART441, UART442, UART443, UART444, UART445, UART446, UART447, UART448, UART449, UART450, UART451, UART452, UART453, UART454, UART455, UART456, UART457, UART458, UART459, UART460, UART461, UART462, UART463, UART464, UART465, UART466, UART467, UART468, UART469, UART470, UART471, UART472, UART473, UART474, UART475, UART476, UART477, UART478, UART479, UART480, UART481, UART482, UART483, UART484, UART485, UART486, UART487, UART488, UART489, UART490, UART491, UART492, UART493, UART494, UART495, UART496, UART497, UART498, UART499, UART500, UART501, UART502, UART503, UART504, UART505, UART506, UART507, UART508, UART509, UART510, UART511, UART512, UART513, UART514, UART515, UART516, UART517, UART518, UART519, UART520, UART521, UART522, UART523, UART524, UART525, UART526, UART527, UART528, UART529, UART530, UART531, UART532, UART533, UART534, UART535, UART536, UART537, UART538, UART539, UART540, UART541, UART542, UART543, UART544, UART545, UART546, UART547, UART548, UART549, UART550, UART551, UART552, UART553, UART554, UART555, UART556, UART557, UART558, UART559, UART560, UART561, UART562, UART563, UART564, UART565, UART566, UART567, UART568, UART569, UART570, UART571, UART572, UART573, UART574, UART575, UART576, UART577, UART578, UART579, UART580, UART581, UART582, UART583, UART584, UART585, UART586, UART587, UART588, UART589, UART590, UART591, UART592, UART593, UART594, UART595, UART596, UART597, UART598, UART599, UART600, UART601, UART602, UART603, UART604, UART605, UART606, UART607, UART608, UART609, UART610, UART611, UART612, UART613, UART614, UART615, UART616, UART617, UART618, UART619, UART620, UART621, UART622, UART623, UART624, UART625, UART626, UART627, UART628, UART629, UART630, UART631, UART632, UART633, UART634, UART635, UART636, UART637, UART638, UART639, UART640, UART641, UART642, UART643, UART644, UART645, UART646, UART647, UART648, UART649, UART650, UART651, UART652, UART653, UART654, UART655, UART656, UART657, UART658, UART659, UART660, UART661, UART662, UART663, UART664, UART665, UART666, UART667, UART668, UART669, UART670, UART671, UART672, UART673, UART674, UART675, UART676, UART677, UART678, UART679, UART680, UART681, UART682, UART683, UART684, UART685, UART686, UART687, UART688, UART689, UART690, UART691, UART692, UART693, UART694, UART695, UART696, UART697, UART698, UART699, UART700, UART701, UART702, UART703, UART704, UART705, UART706, UART707, UART708, UART709, UART710, UART711, UART712, UART713, UART714, UART715, UART716, UART717, UART718, UART719, UART720, UART721, UART722, UART723, UART724, UART725, UART726, UART727, UART728, UART729, UART730, UART731, UART732, UART733, UART734, UART735, UART736, UART737, UART738, UART739, UART740, UART741, UART742, UART743, UART744, UART745, UART746, UART747, UART7

- 
- The screenshot displays the STM32CubeMX software interface for configuring the STM32F303RETx NUCLEO-F303RE. The 'Pinout & Configuration' tab is selected, showing the 'CAN Mode and Configuration' section. The 'Master Mode' is selected. The 'Configuration' section shows the 'Reset Configuration' button and the 'User Constants', 'NVIC Settings', and 'GPIO Settings' tabs. The 'Bit Timings Parameters' section is expanded, showing the following values:
- Prescaler (for Time Quantum): 16
  - Time Quantum: 444.4444444444444 ns
  - Time Quanta in Bit Segment 1: 1 Time
  - Time Quanta in Bit Segment 2: 1 Time
  - ReSynchronization Jump Width: 1 Time
- The 'Pinout view' on the right shows the STM32F303RETx LQFP44 package with pins labeled. The CAN\_TX and CAN\_RX pins are highlighted in orange.

- 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):

$$f_{\text{bit}} = f_{\text{can}} / (N * K) = 0.125 \text{ Mbps}$$

This is possible by choosing N = 9, K = 32  $\rightarrow f_{\text{bit}} = 36 \text{ MHz} / (9 * 32) = 0.125$

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:

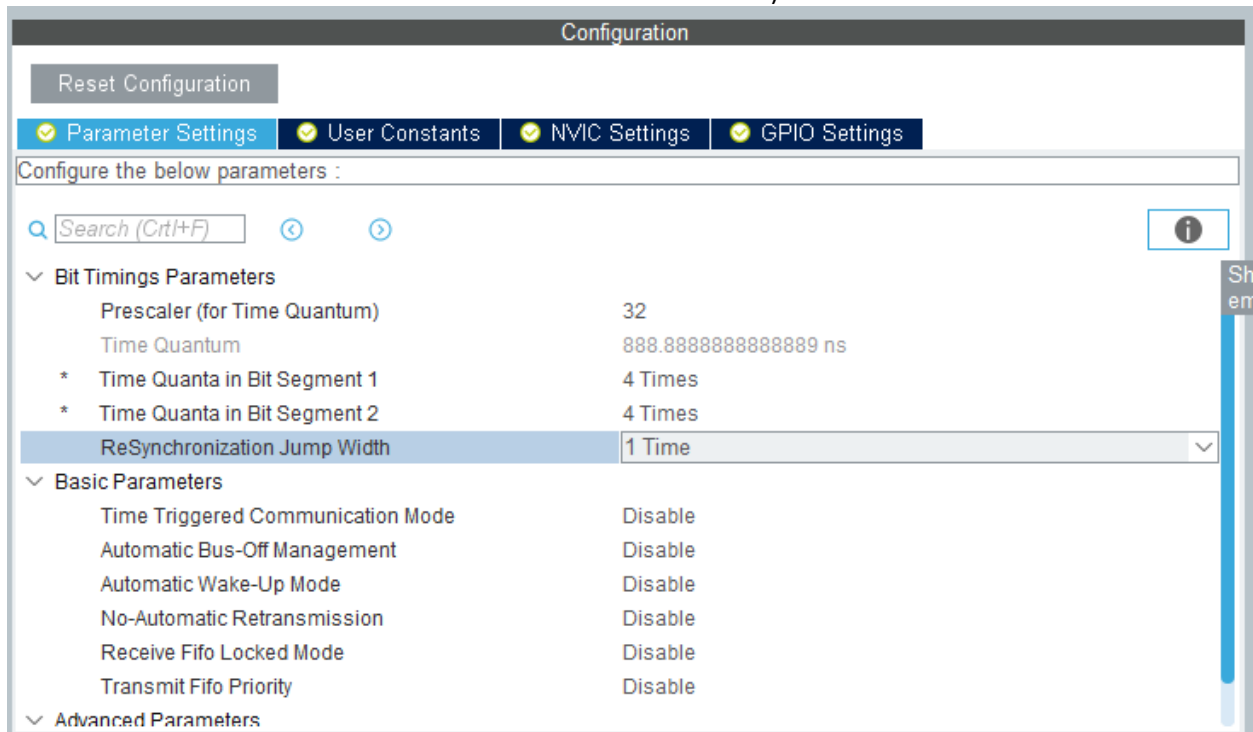
K == 'Prescaler (for time quantum)'

SYNC == 'ReSynchronization jump width'

TSEG1 == 'Time quanta in bit segment 1'

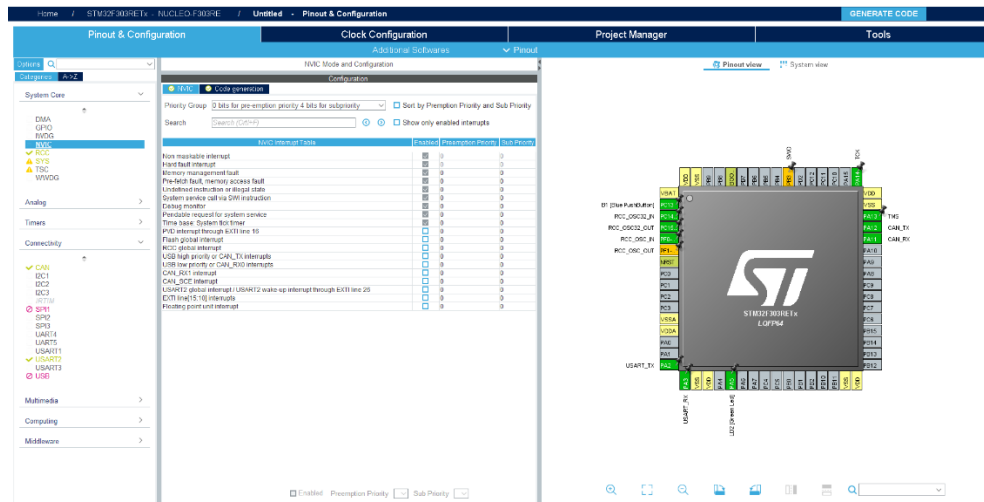
TSEG2 == 'Time quanta in bit segment 2'

N == SYNC + TSEG1 + TSEG2 (not set directly in the configuration but is the sum of the above three values)



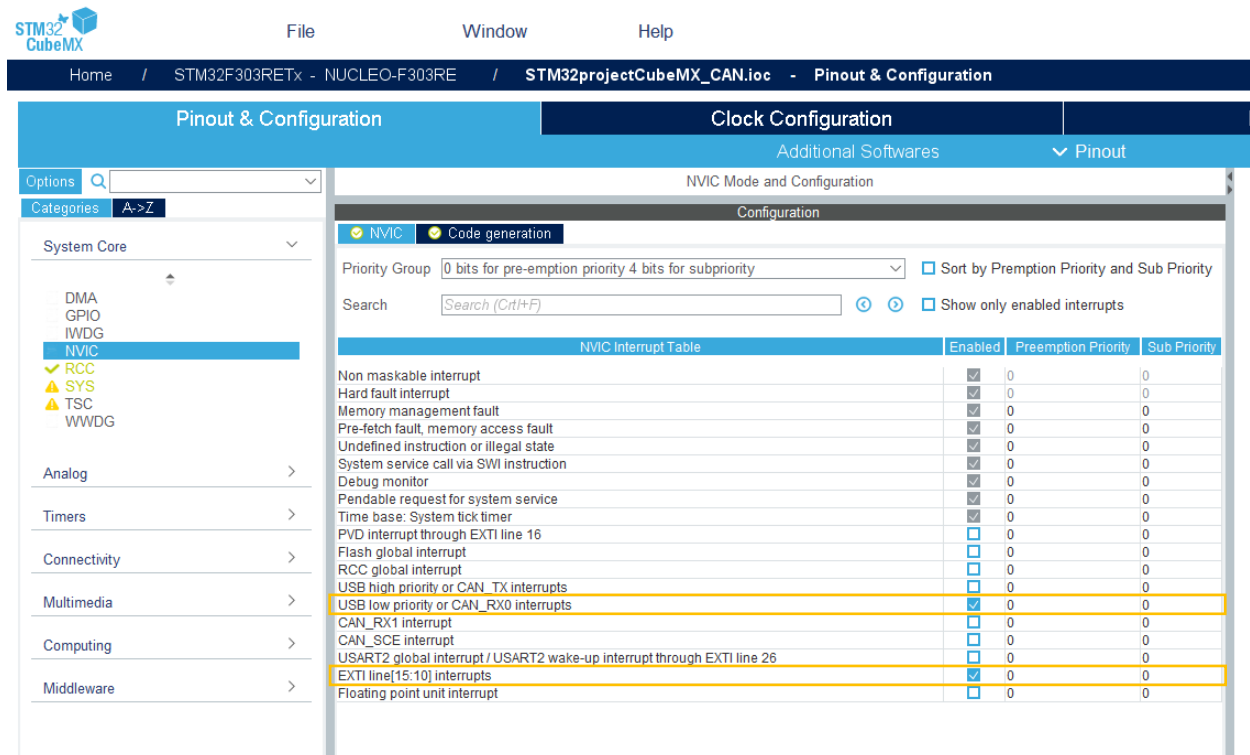
(Leave all other parameters to their defaults – *Disable* and *Normal*)

- 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

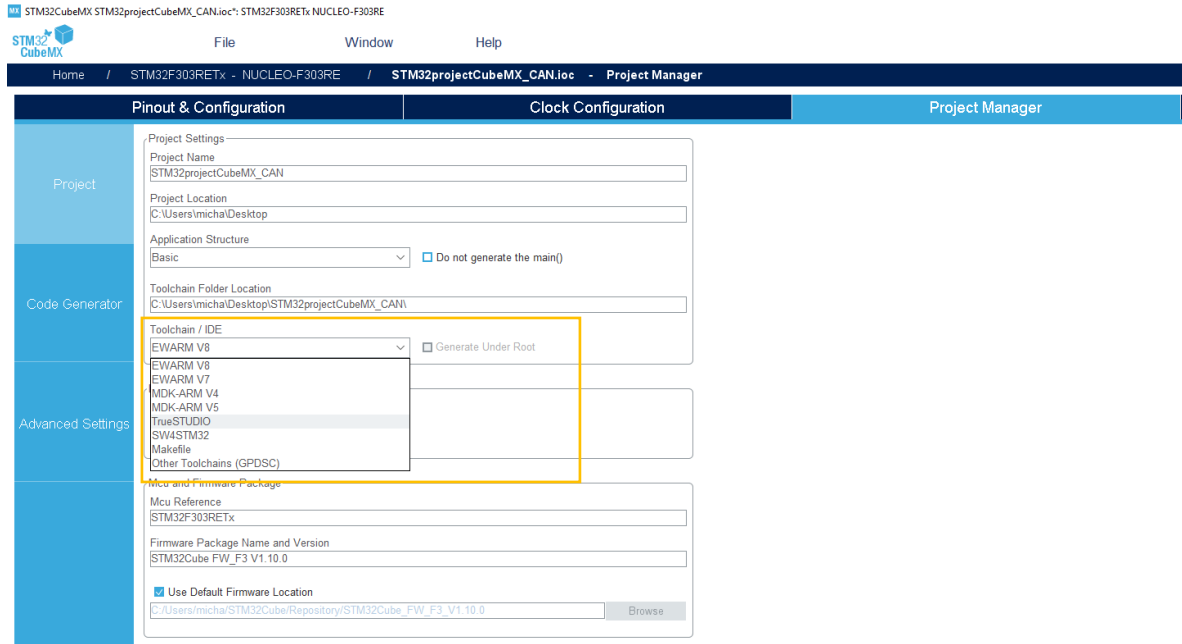


- On the NVIC tab, click on the 'Enable' check box for 'USB low priority or CAN\_RX0 interrupts' (USB-CAN Rx interrupts are shared between CAN and USB). This enables the Nested Vector Interrupt Controller (NVIC) for CAN Rx messages. Also on the NVIC tab, click on the 'Enable' check box for EXTI line[15:10] interrupts. This enables the GPIO interrupts (e.g. for the pushbutton(s)).

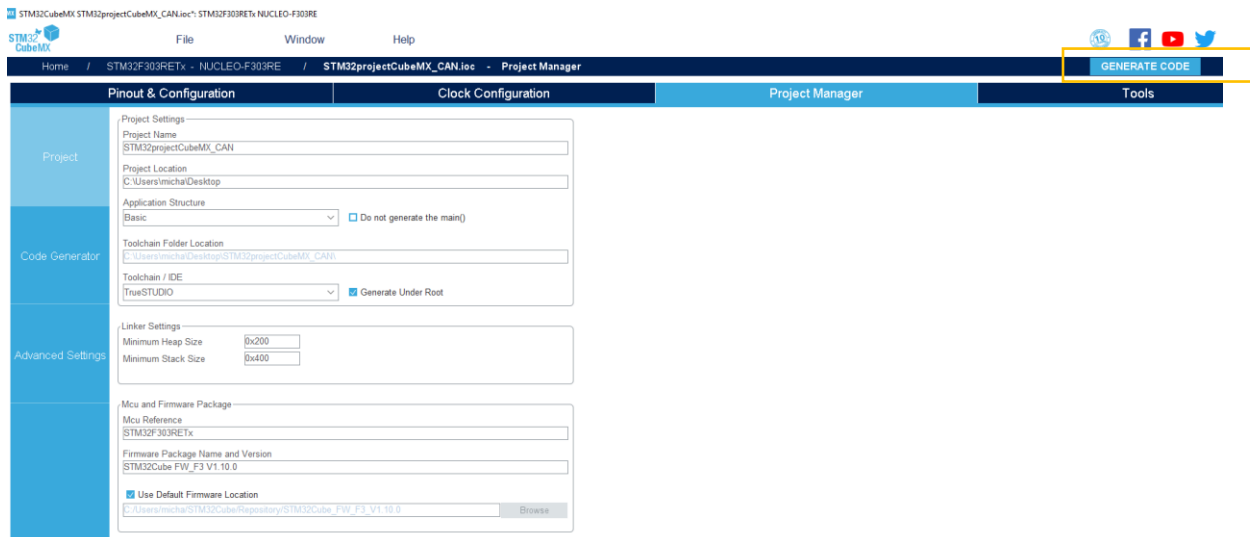
STM32CubeMX STM32projectCubeMX\_CAN.ioc STM32F303RETx NUCLEO-F303RE



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)



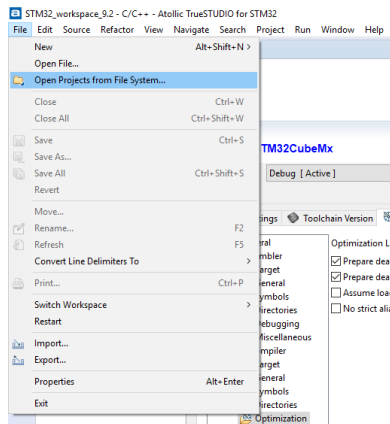
- c. Select the location and name for the deployed TrueStudio project to deploy the base initialization code (save the project).
  - d. Click on Generate Code



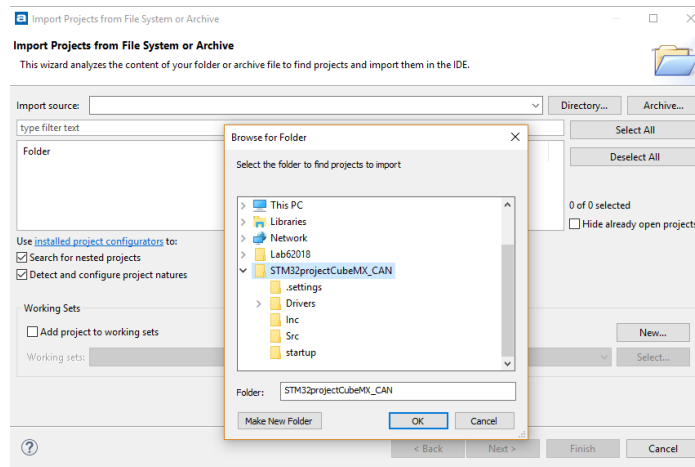
- e. Project files will be created in the saved directory (note the .ioc file is the STM32CubeMX project, you can use this later to regenerate the project and add new functionality)
  - f. Close STM32CubeMx generator

## 8. Open Atollic TrueStudio

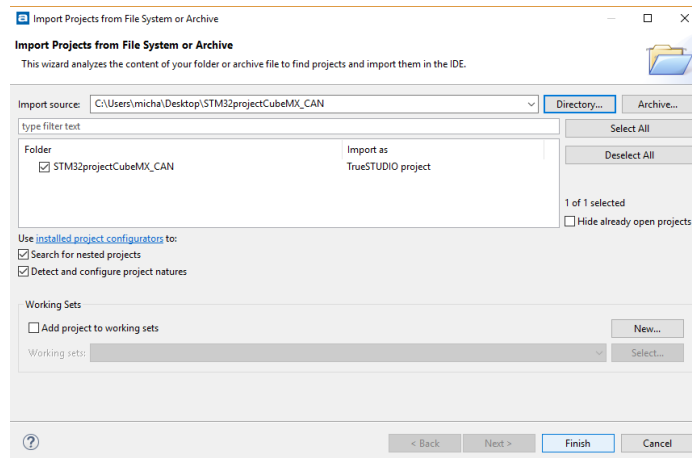
- Open the Atollic TrueStudio project generated by STM32CubeMX by clicking on File >> Open Projects from File System



- Select 'Directory' and choose the folder you created the TrueStudio Project in

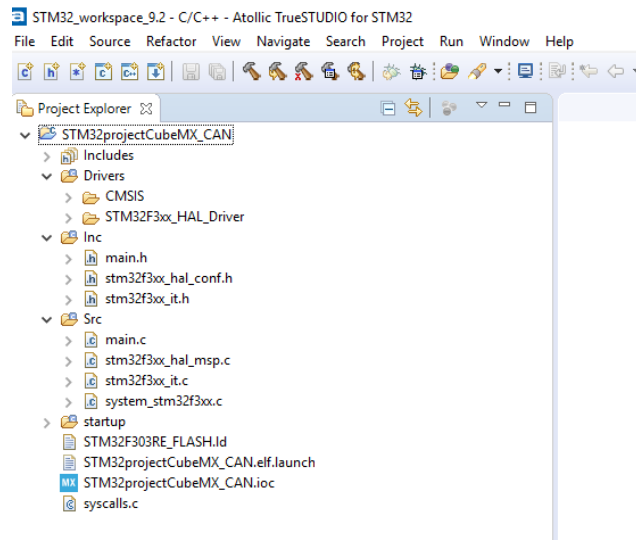


- Hit 'Finish' to open the project

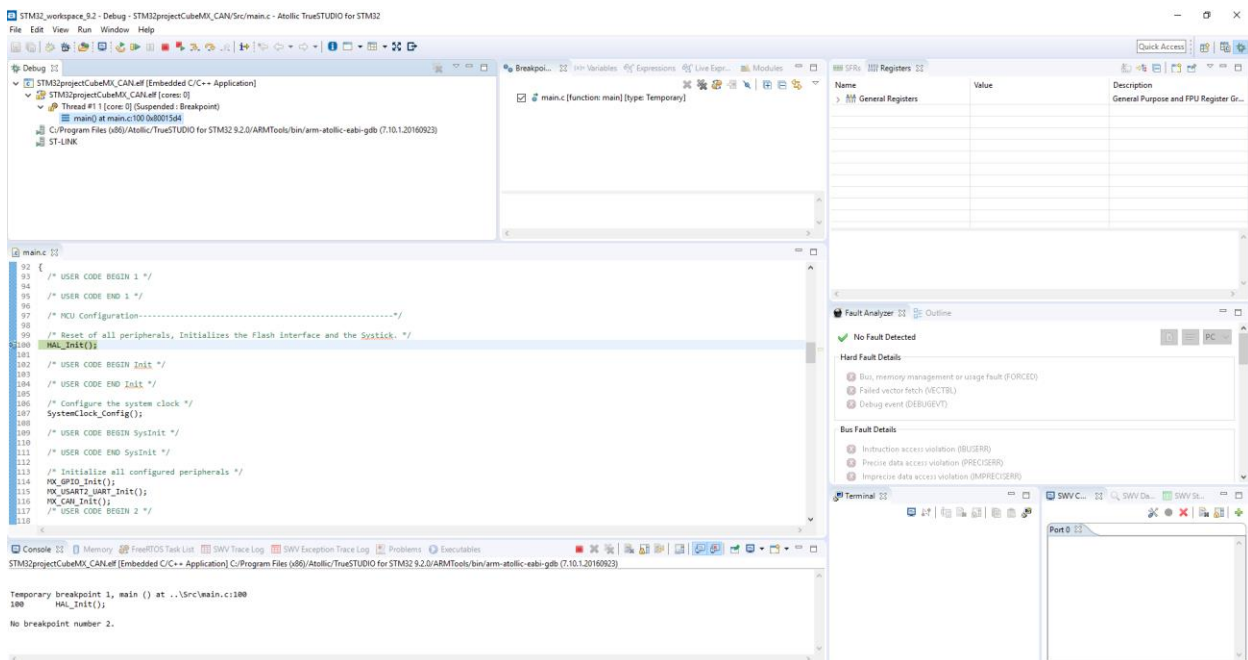




- d. Close the 'Information center' tab to show the newly opened project
- e. Project files will be loaded into Atollic TrueStudio for you to edit/add

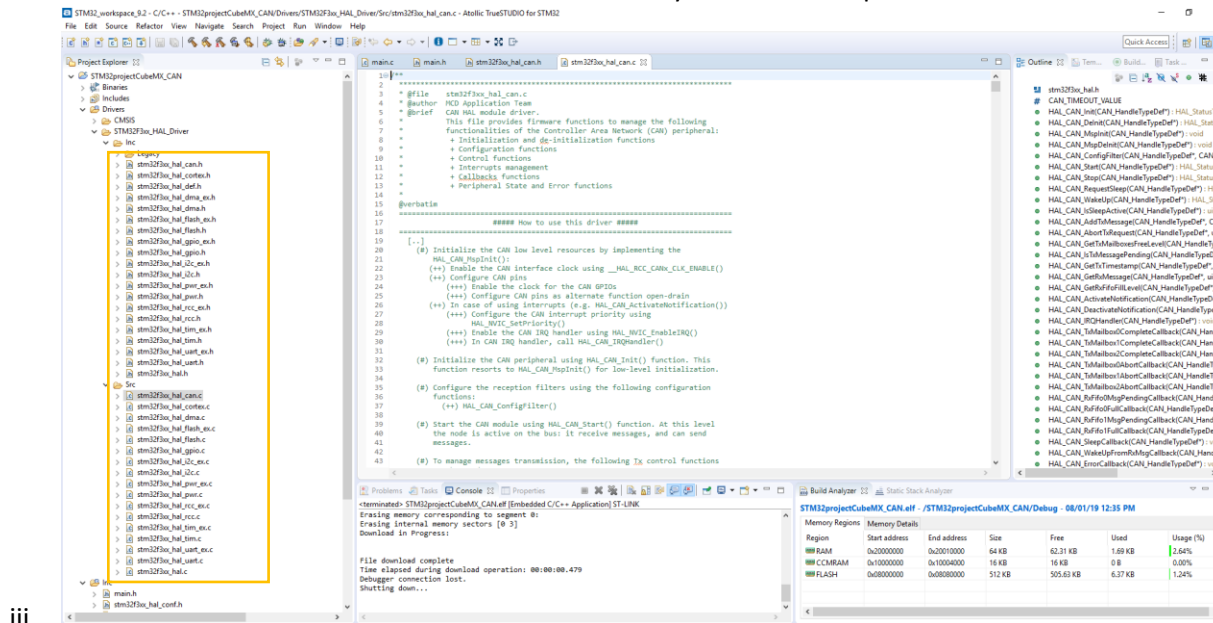


9. Test the default code to make sure it can be deployed to the STM32
  - a. Connect to the STM32 using the USB cable
  - b. Note: You may need to update ST-Link within Atollic TrueStudio and/or the Firmware on your STM32F303RE Board (it will ask you and update automatically, if required)
  - c. Click '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'

- 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).
- 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.



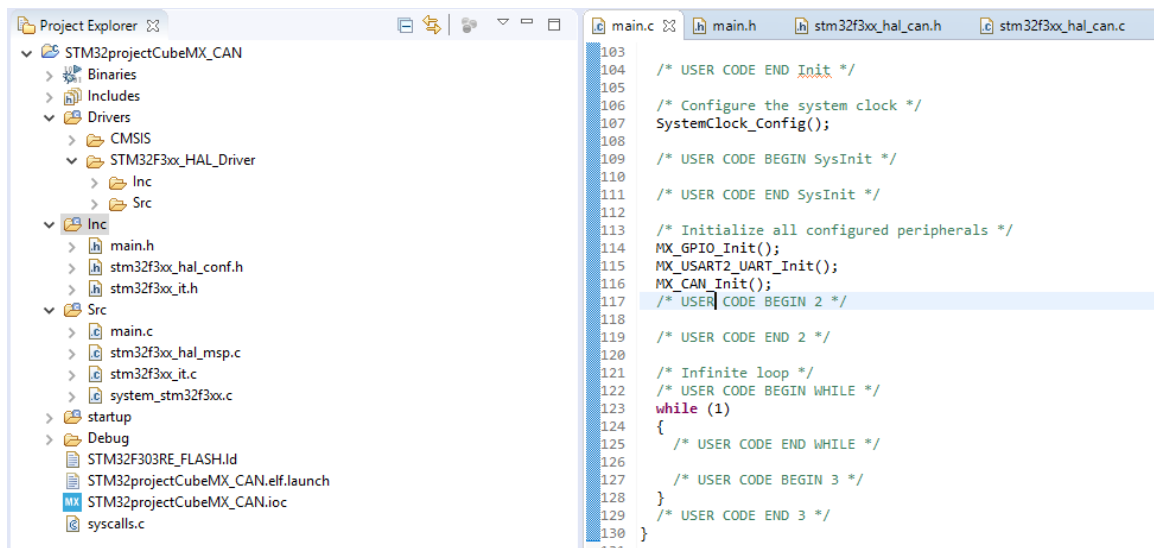
iii.

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.

```

54 /* Private define -----*/
55 /* USER CODE BEGIN PD */
56 #define ID 0x0100 // ID of supervisory controller - Change this depending on floor #
57 #define GO_TO_FLOOR_1 0x05 // Floor 1
58 #define GO_TO_FLOOR_2 0x06 // Floor 2
59 #define GO_TO_FLOOR_3 0x07 // Floor 3
60 #define NO_BUTTON_PRESSED 0 // Default value of the BUTTON flag - no button pressed
61 #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

```

75 /* USER CODE BEGIN PV */
76 CAN_TxHeaderTypeDef TxHeader;
77 CAN_RxHeaderTypeDef RxHeader;
78 uint8_t TxData[8];
79 uint8_t RxData[8];
80 uint32_t TxMailbox;
81
82 uint8_t msg = GO_TO_FLOOR_1; // Message is 'Go to Floor 1'
83 uint8_t BUTTON = NO_BUTTON_PRESSED; // Button pressed flag (value changed in callback function for given interrupt)
84
85 /* USER CODE END PV */

```

**CAN\_TxHeaderTypeDef** and **CAN\_RxHeaderTypeDef** 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.

```
137 /* Infinite loop */
138 /* USER CODE BEGIN WHILE */
139 while (1)
140 {
141     /* USER CODE END WHILE */
142
143     /* USER CODE BEGIN 3 */
144     uint8_t i;
145     // Receive
146     if (RxData[0] == GO_TO_FLOOR_1) {
147         HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);           // Turn on LED2
148         HAL_Delay(2000);                                   // Keep LED on for 2 seconds
149         for (i=0; i<8; i++) {
150             RxData[i] = 0x00;                             // Reset the RxData[] buffer (used as flag)
151         }
152         HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);           // Turn off LED2
153         HAL_Delay(100);                                    // Need a delay after toggle
154     }
155
156     // Transmit
157     if (BUTTON != 0) {
158         if (BUTTON == BLUE_BUTTON_PRESSED) {
159             HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);       // Blue button pressed --> Turn on LED2 for 2 seconds and Transmit message
160             HAL_Delay(2000);                               // Turn on LED2
161             TxData[0] = msg;                               // Leave it on for 2 seconds
162             if (HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox) != HAL_OK) { // Store the 1 character message to transmit into the TxData buffer and transmit over the CAN b
163                 Error_Handler();                          // Transmit the message
164             }                                              // Transmission error
165             HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);       // Turn off LED2
166             BUTTON = NO_BUTTON_PRESSED;                  // Reset the BUTTON flag
167         }
168     }
169 }
170 /* USER CODE END 3 */
171 }
```

## 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 - don't 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();
    }
}
```

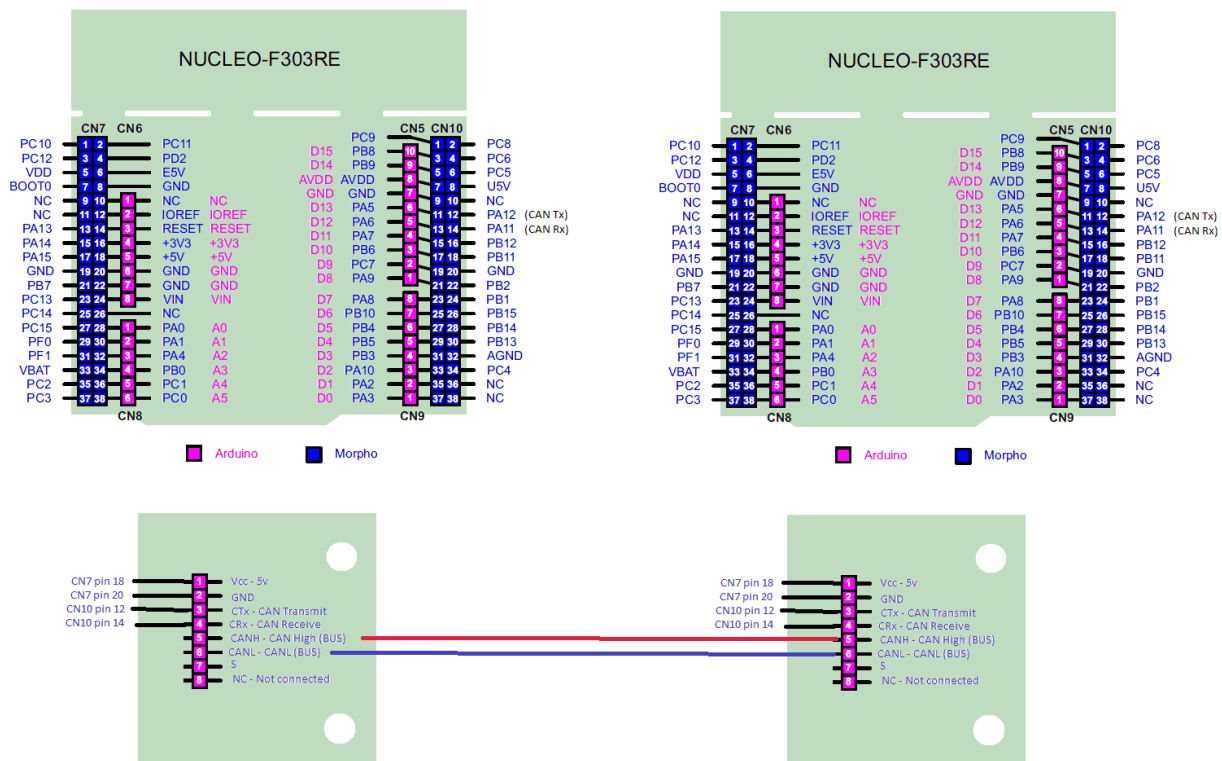
The callback function calls HAL\_CAN\_GetRxMessage() to store the received message in RxData[].

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
    }
}
```

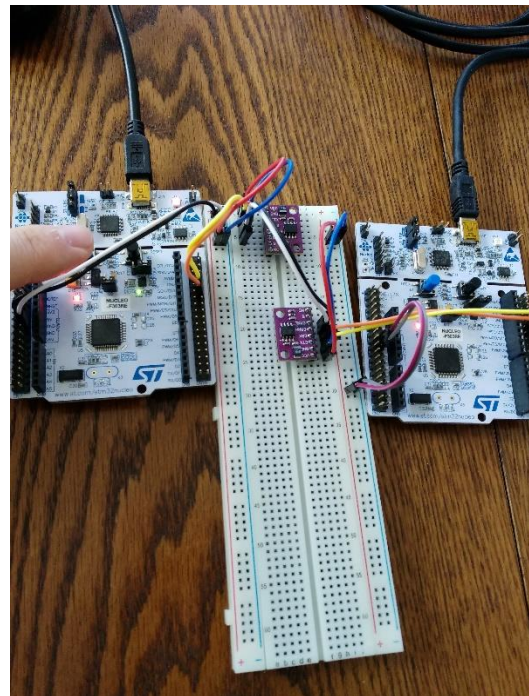
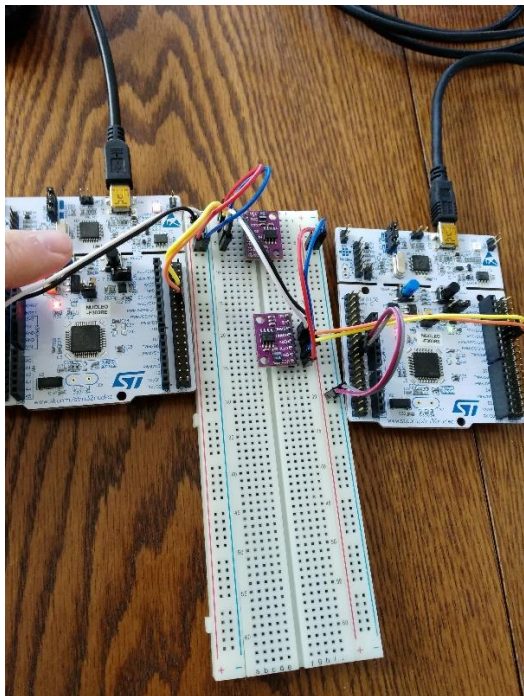
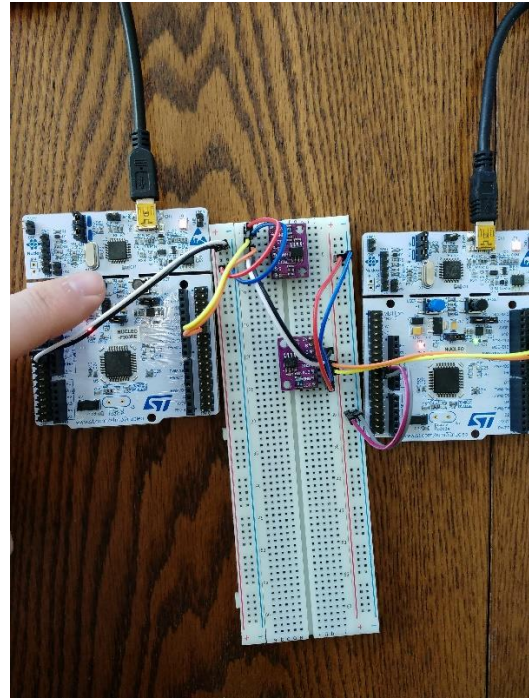
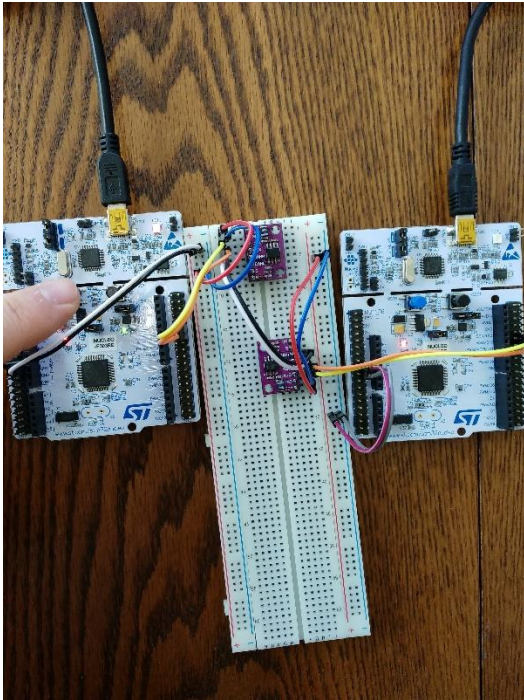
When the BUTTON flag is changed to BLUE\_BUTTON\_PRESSED the required action (turn on LED and transmit message) it is handled in main().

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 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](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)