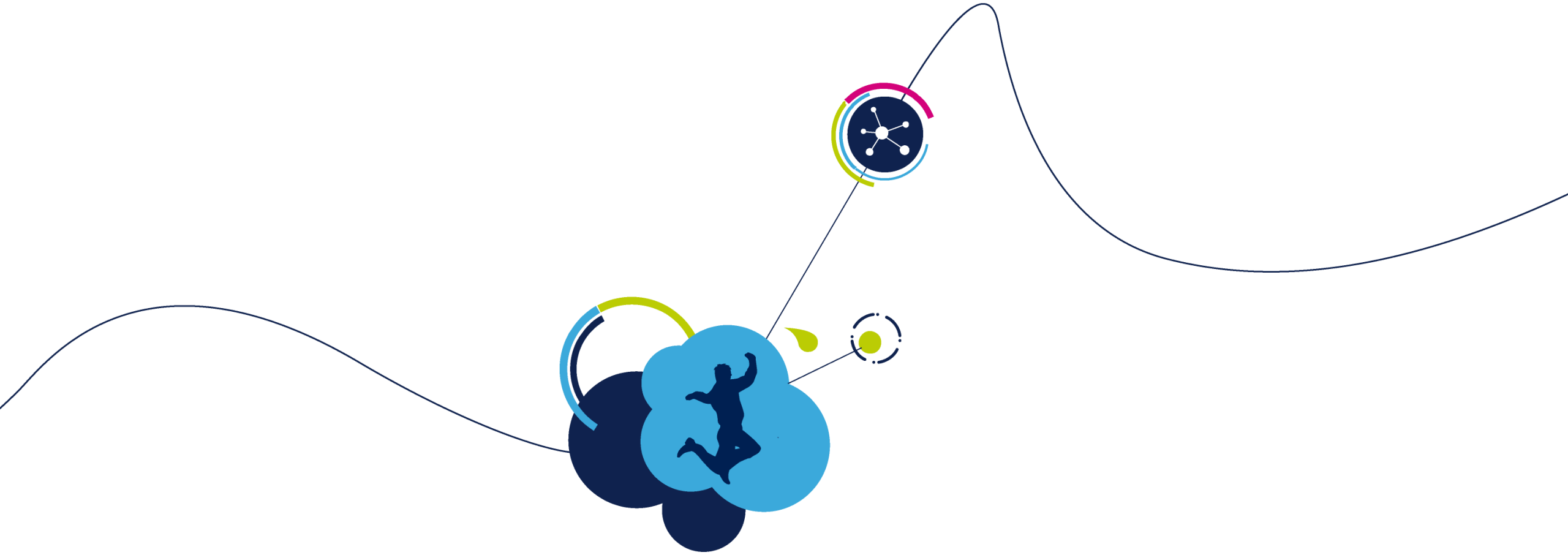
STM32L4 technical training

Controller Area Network (CAN)

Hands-on session

life.augmented



CAN Lab

CAN connectivity – sending and receiving of CAN messages

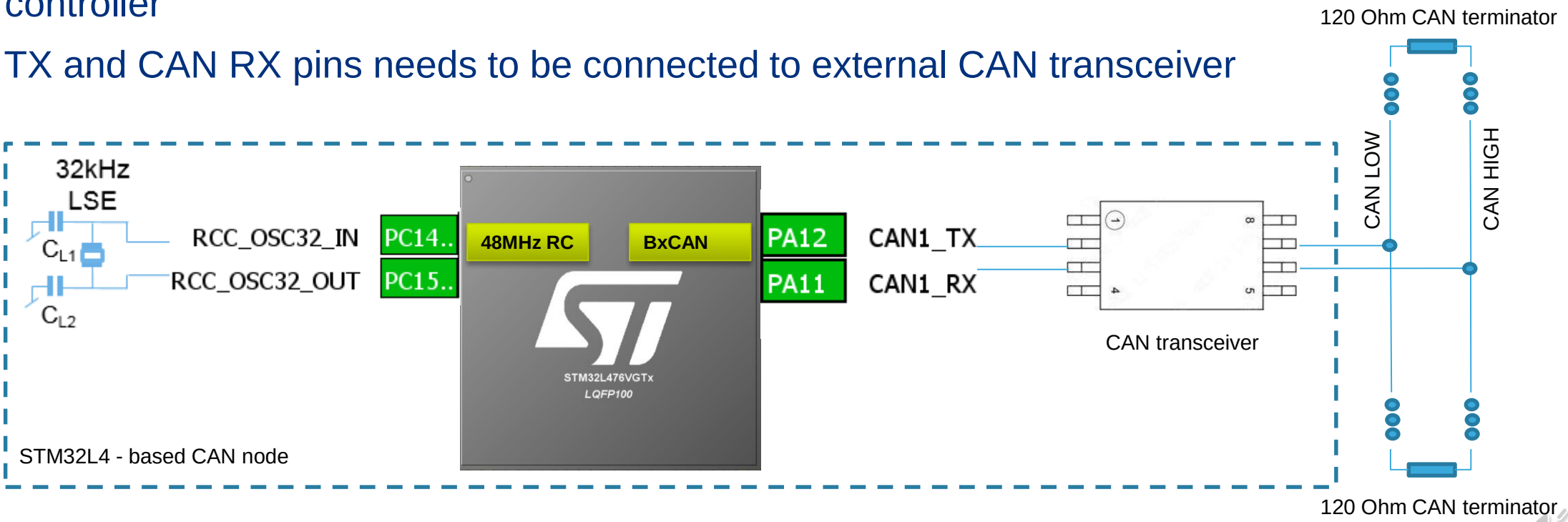
life.augmented

CAN connectivity

- Objective
  - Learn how to configure CAN in CubeMX
  - Learn how to generate code in CubeMX and use HAL functions
- Method
  - Develop an application, which sends CAN messages and receives CAN messages

CAN hardware connection

- STM32L4 as a CAN node
  - BxCAN - embedded CAN controller 2.0A/B
  - Internal RC 48 MHz (MSI – Multi Speed Internal), which can be trimming by LSE (Low Speed External) and can be used as an accurate source of clock signal for CAN controller
  - CAN TX and CAN RX pins needs to be connected to external CAN transceiver



life.augmented

CAN connection

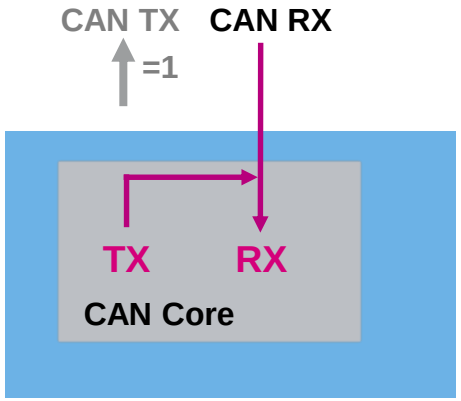
STM32L476RG-Discovery

- STM32L476RG-Discovery is not equipped with a CAN transceiver, which is needed for CAN connectivity in network.

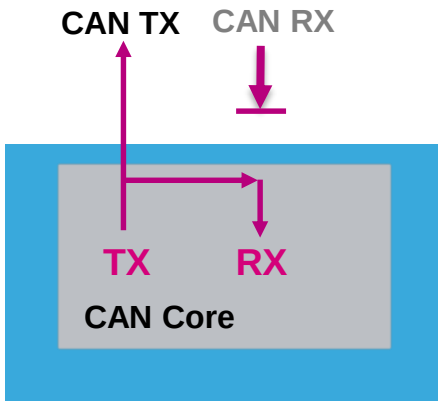
Not ready for CAN network connectivity

- For purpose of CAN evaluation with STM32L476RG-Discovery, one of CAN test modes can be used (silent, loopback or silent+loopback).

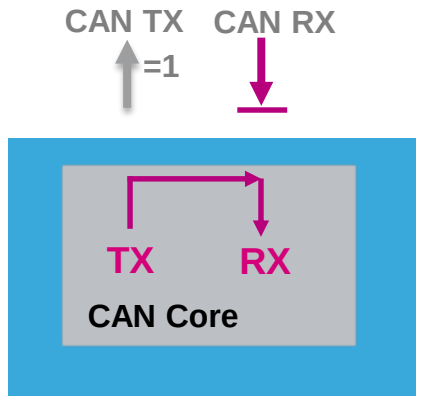
1. Silent



2. Loopback



3. Combined Loopback and Silent

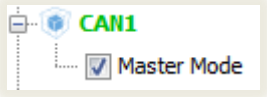


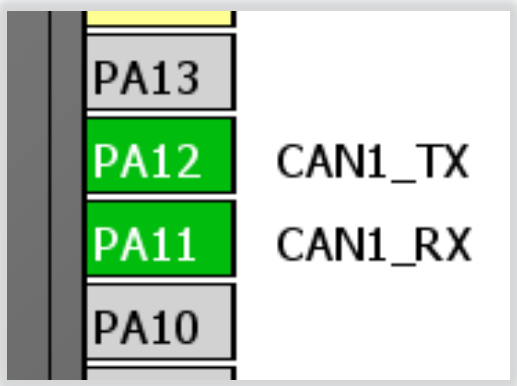
STM32CubeMX

Selecting CAN interface and clock

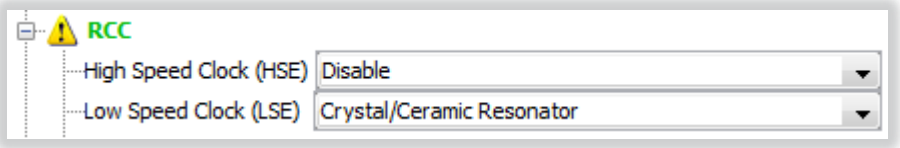
- Create project in STM32CubeMX
  - Menu > File > New Project
  - Select STM32L4 -> STM32L4x6 -> LQFP100 package -> STM32L476VGTx

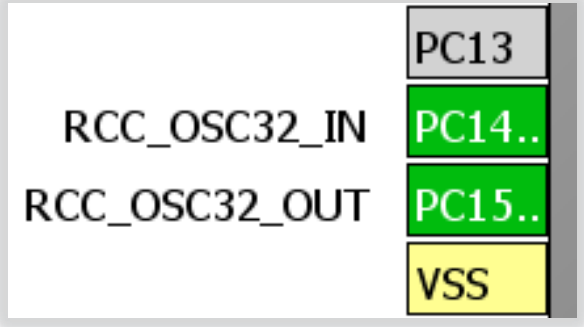
- Select CAN:
  - Select “**Master Mode**” for **CAN1**





- Select LSE:
  - Select “**Crystal/Ceramic Resonator**” for **Low Speed Clock (LSE)** of **RCC**





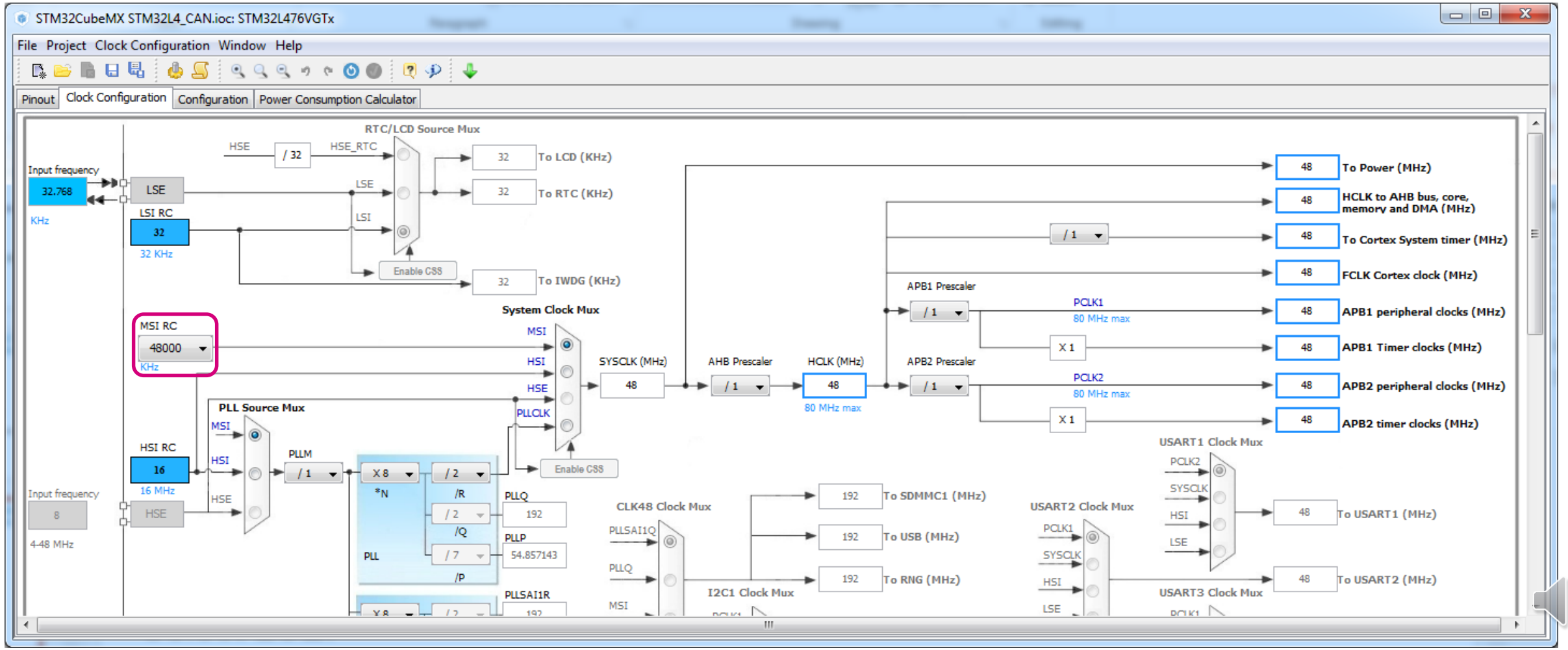
LSE is needed to trimm the MSI

life.augmented

STM32CubeMX

clock configuration

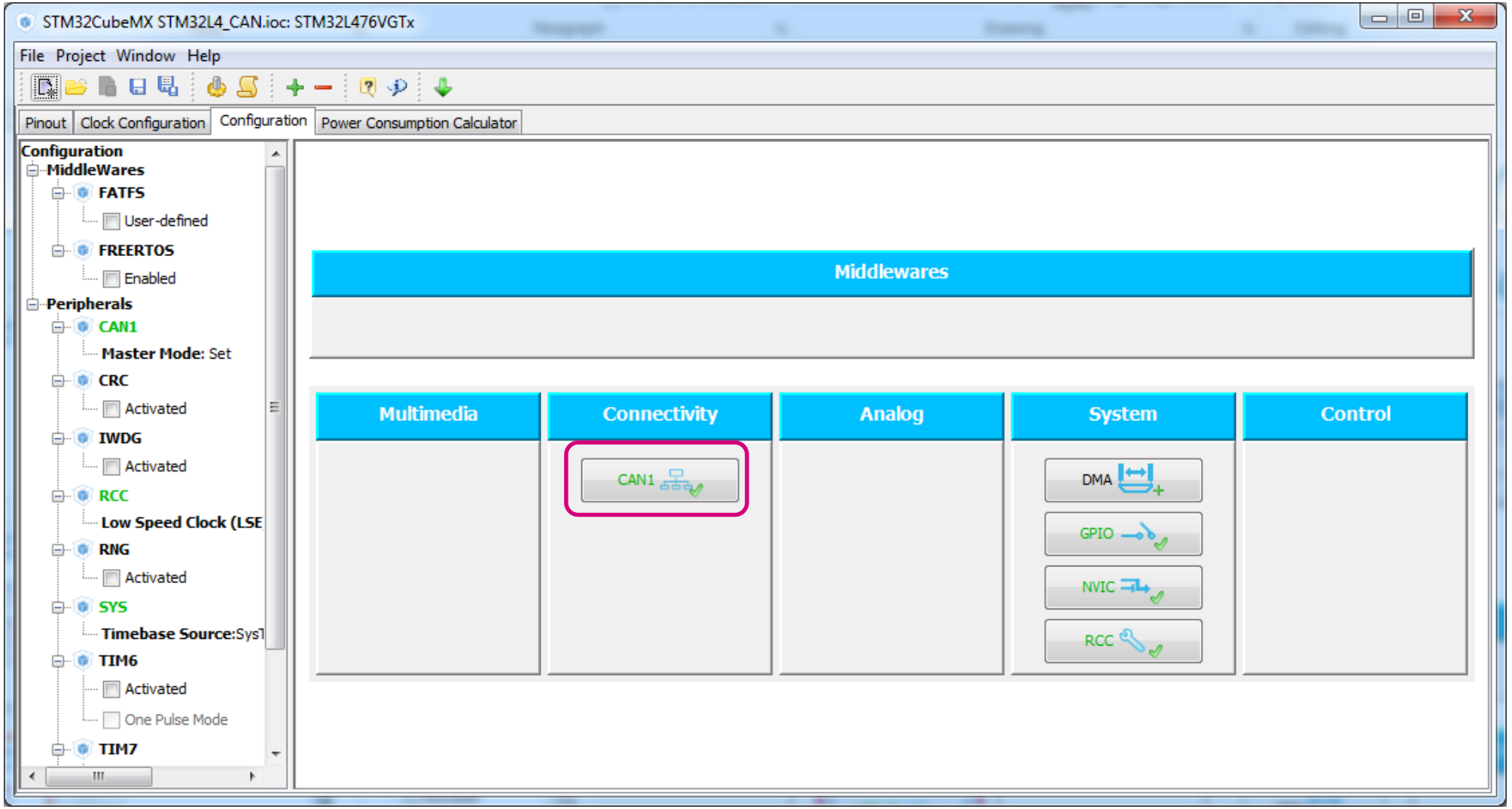
- Go to Clock Configuration tab and configure MCU clock system:
  - Change MSI default value (4 MHz) to 48 MHz



STM32CubeMX

Configure CAN

- Go to Configuration tab and select CAN peripheral

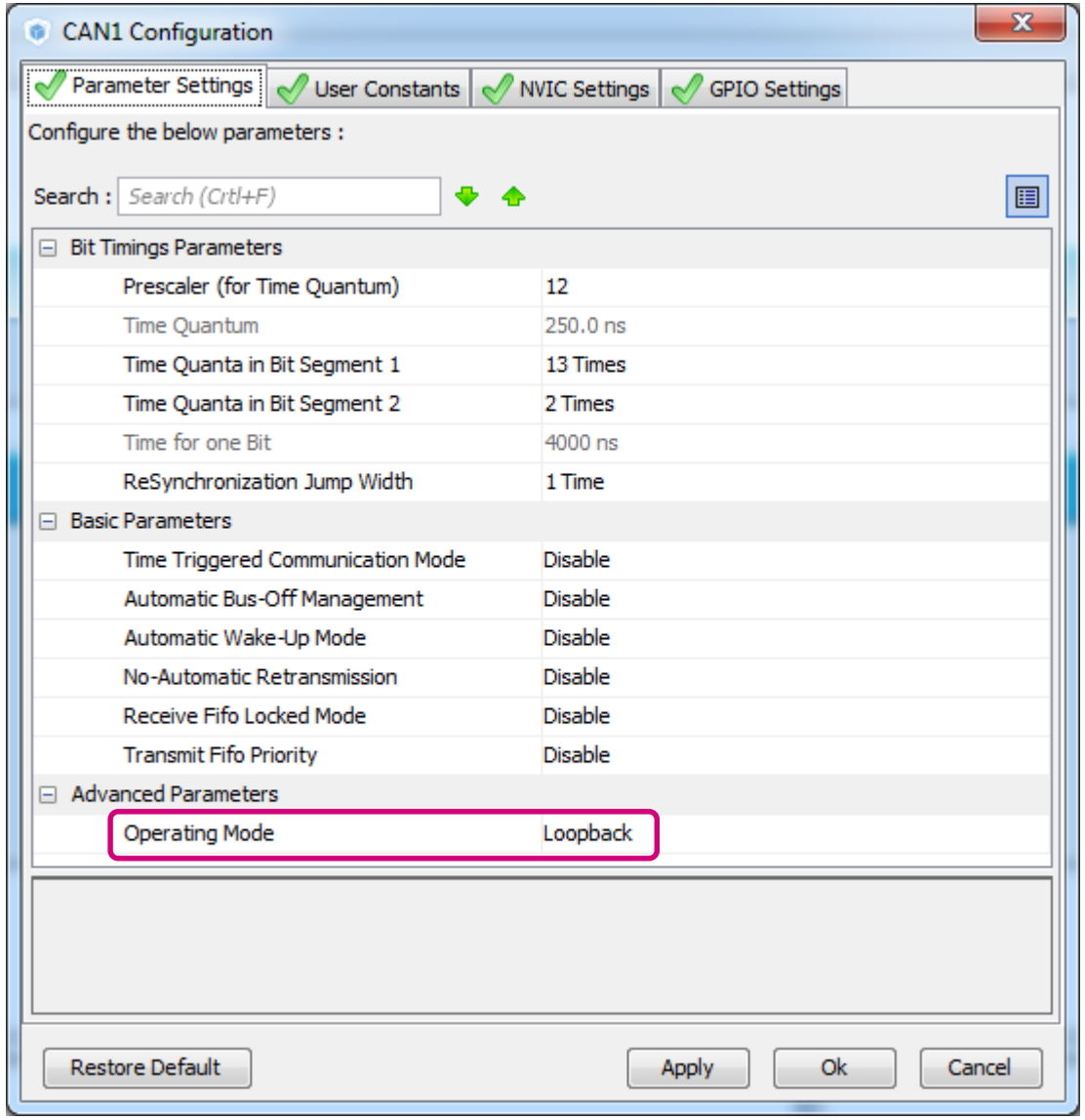


life.augmented

STM32CubeMX

configuration of CAN mode

- Select Parameter Settings tab
  - Change Operating Mode as Loopback or Silent
- Press **Ok** to confirm the configuration

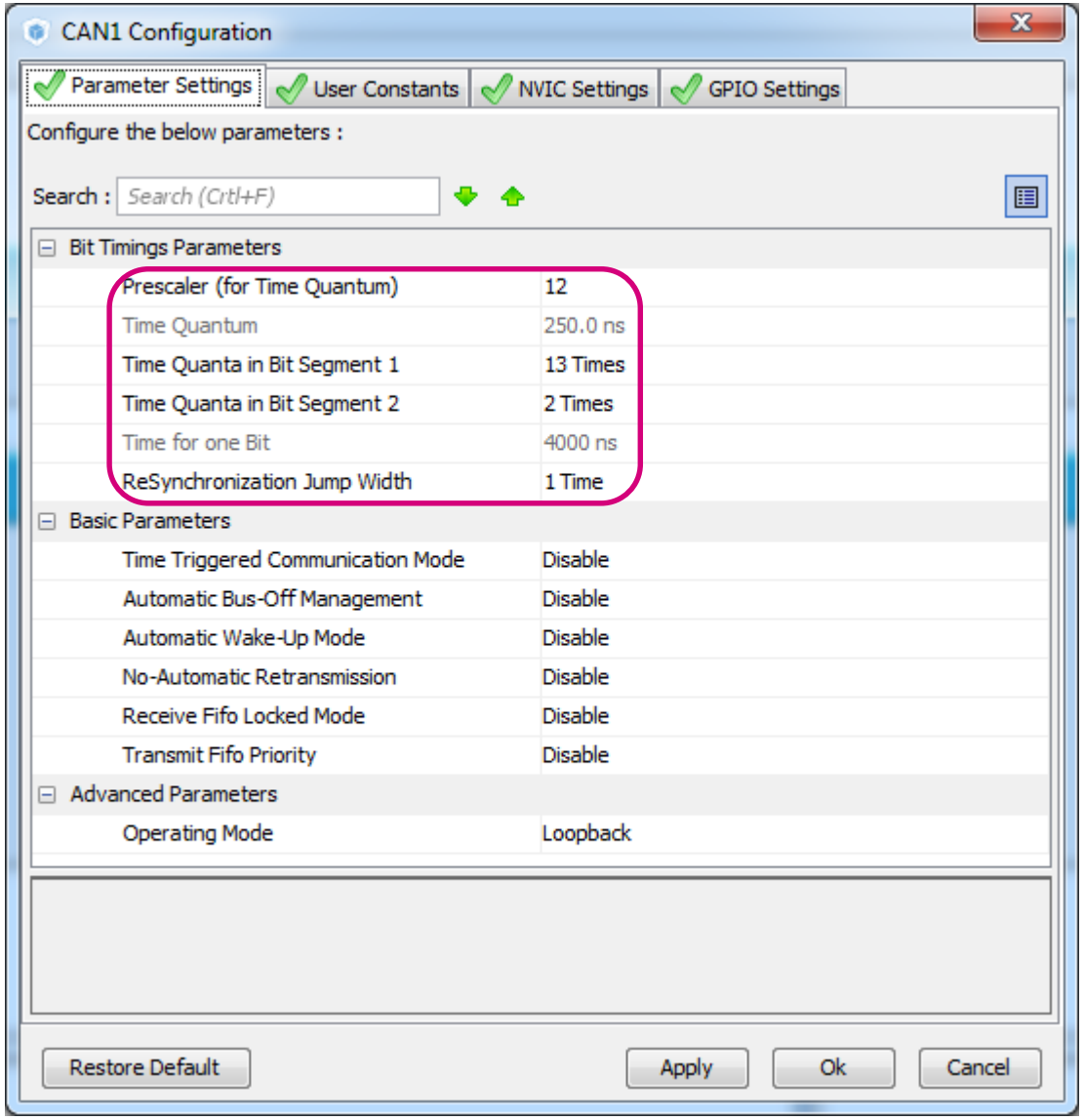


life.augmented

STM32CubeMX

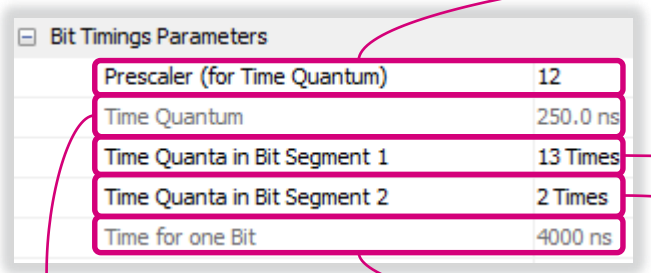
configuration of CAN baudrate

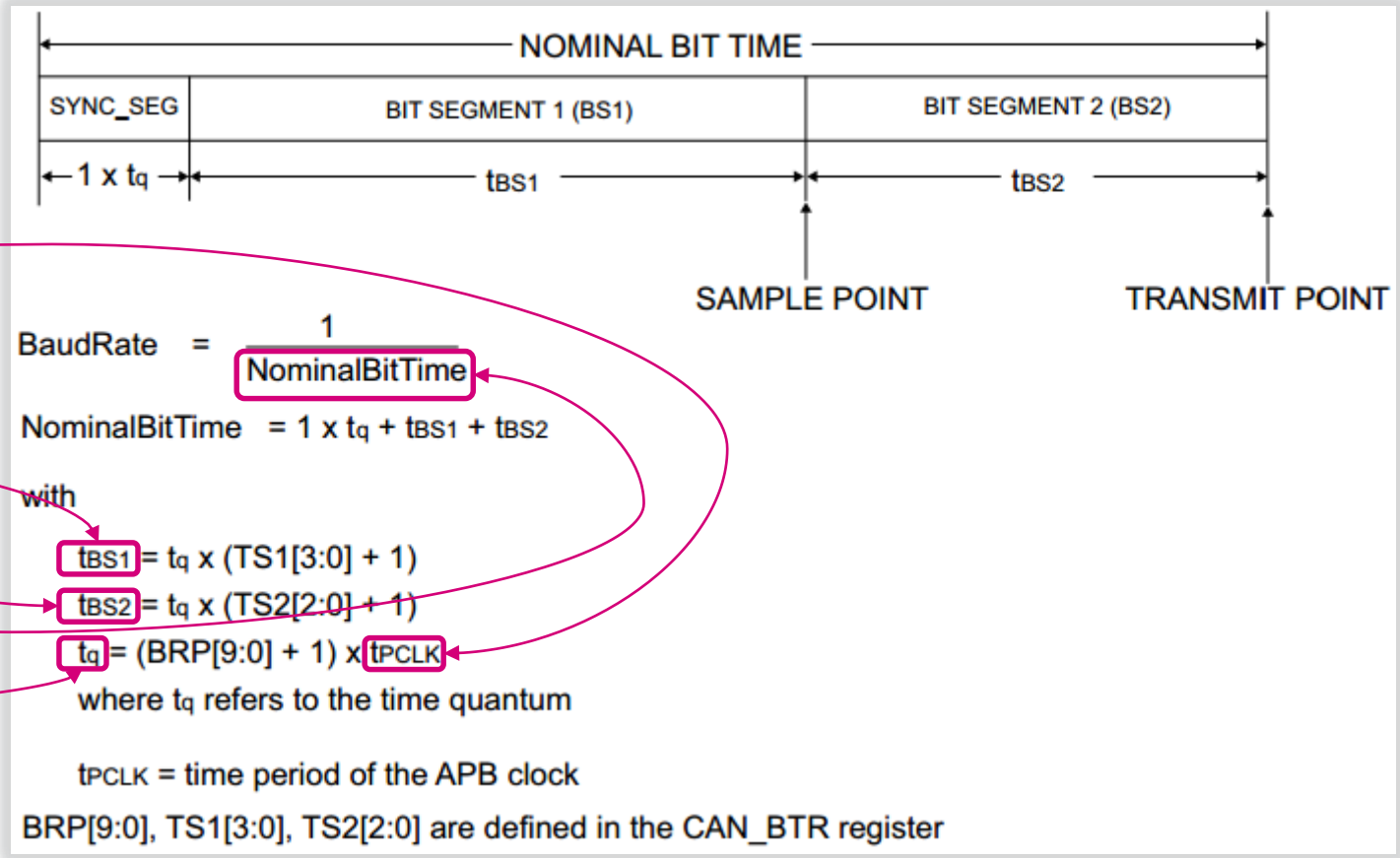
- Select Parameter Settings tab
  - Fill in Bit Timing Parameters to set CAN baudrate
- Press **Ok** to confirm the configuration



How to understand parameters, which have impact on CAN baudrate?

- Formula used to calculate CAN baudrate





- More information:

life.augmented

RM0351
Reference manual

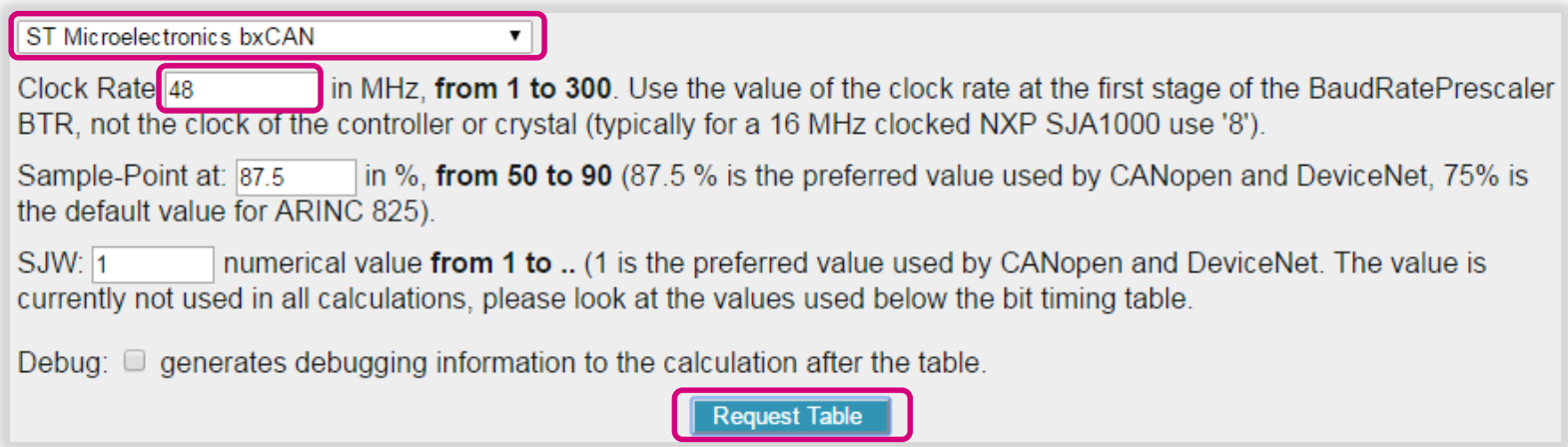
STM32L4x5 and STM32L4x6 advanced ARM®-based 32-bit MCUs

46 Controller area network (bxCAN) 1576

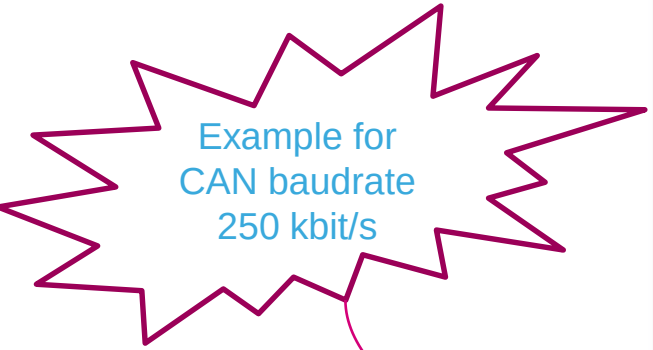
Easy configuration of CAN baudrate

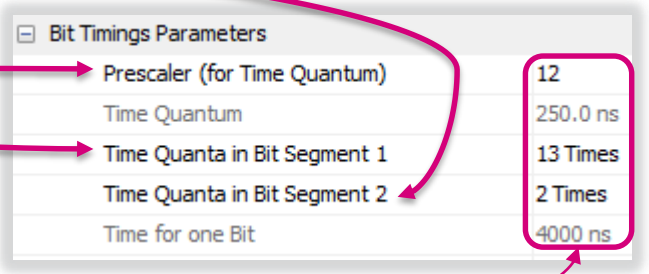
- http://www.bittiming.can-wiki.info/ webpage allows to obtain CAN baudrate configuation parameters automatically

1. Select STMicroelectronics as a CAN controller vendor
2. Select MCU's system clock
3. Click on **Request Table** button
4. Find desired CAN baudrate in the table and copy **clock prescaler**, **SB1** and **SB2** into the CubeMX



Bit Rate	accuracy	Pre-scaler	Number of time quanta	Seg 1 (Prop_Seg+Phase_Seg1)	Seg 2	Sample Point at	Register CAN_BTR
1000	0.0000	3	16	13	2	87.5	0x001c0002
1000	0.0000	4	12	10	1	91.7	0x00090003
1000	0.0000	6	8	6	1	87.5	0x00050005
800	0.0000	4	15	12	2	86.7	0x001b0003
800	0.0000	5	12	10	1	91.7	0x00090004
800	0.0000	6	10	8	1	90.0	0x00070005
500	0.0000	6	16	13	2	87.5	0x001c0005
500	0.0000	8	12	10	1	91.7	0x00090007
500	0.0000	12	8	6	1	87.5	0x0005000b
250	0.0000	12	16	13	2	87.5	0x001c000b



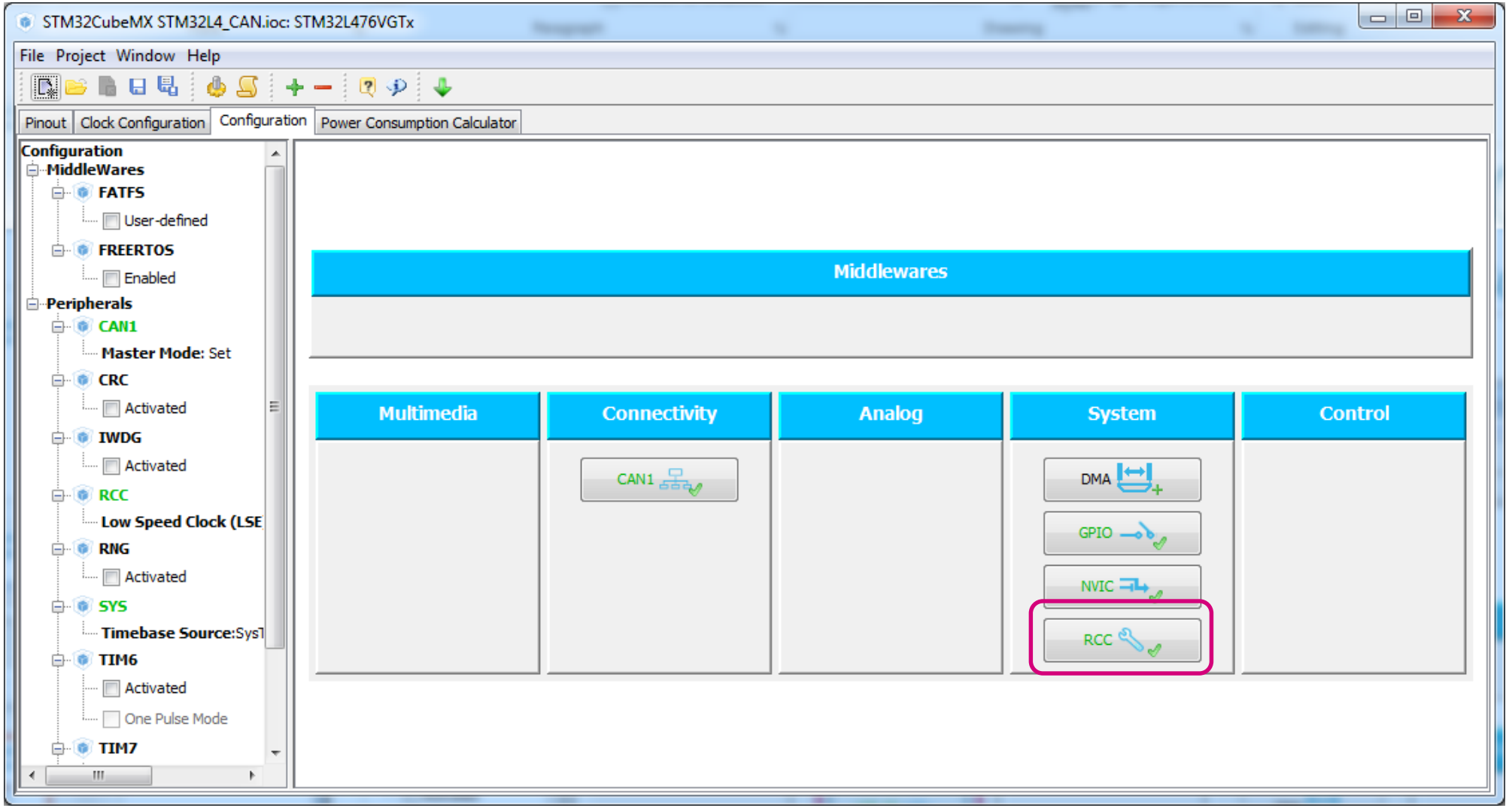


life.augmented

STM32CubeMX

Configure clock

- Go to Configuration tab and select RCC peripheral

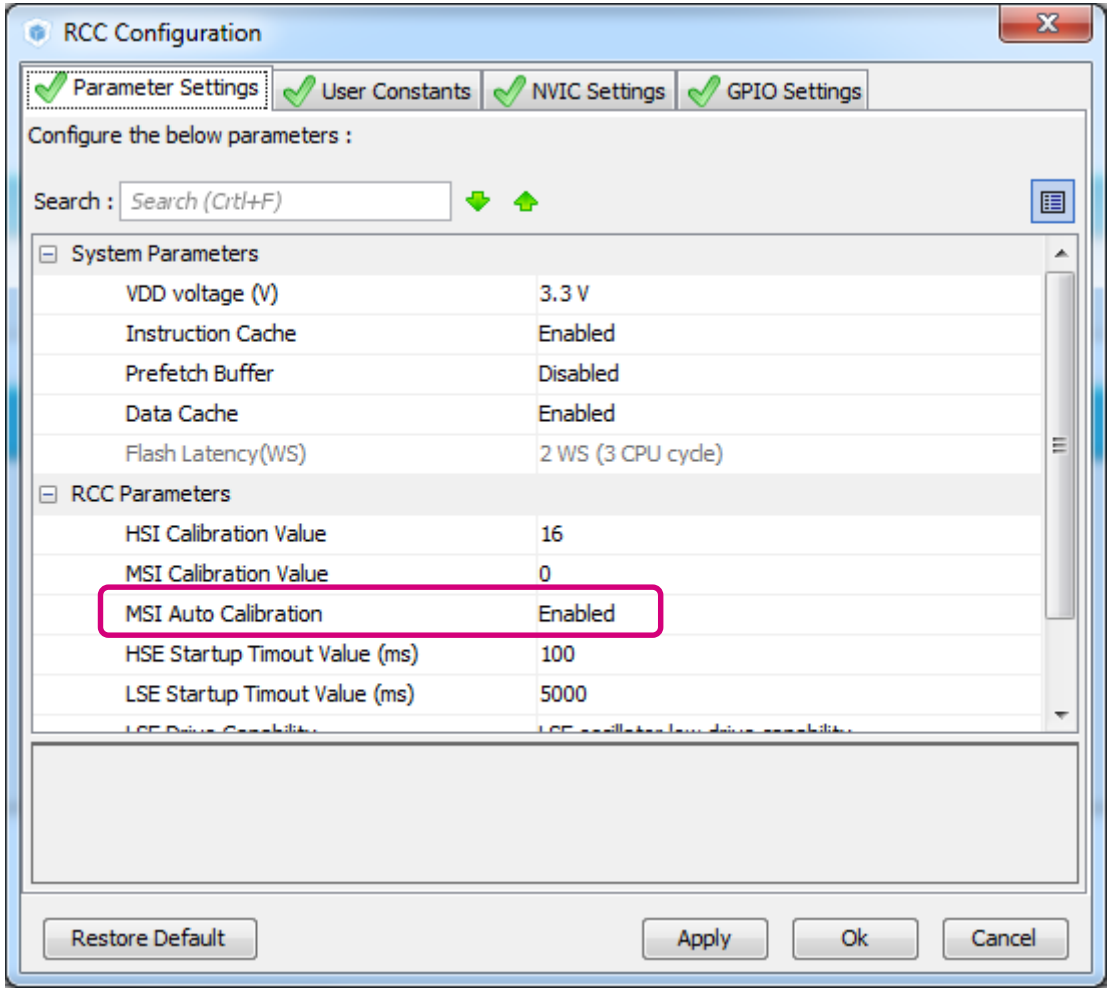


life.augmented

STM32CubeMX

configuration of the MSI calibration with LSE

- Select Parameter Settings tab
  - Enable MSI Auto Calibration
- Press **Ok** to confirm the configuration

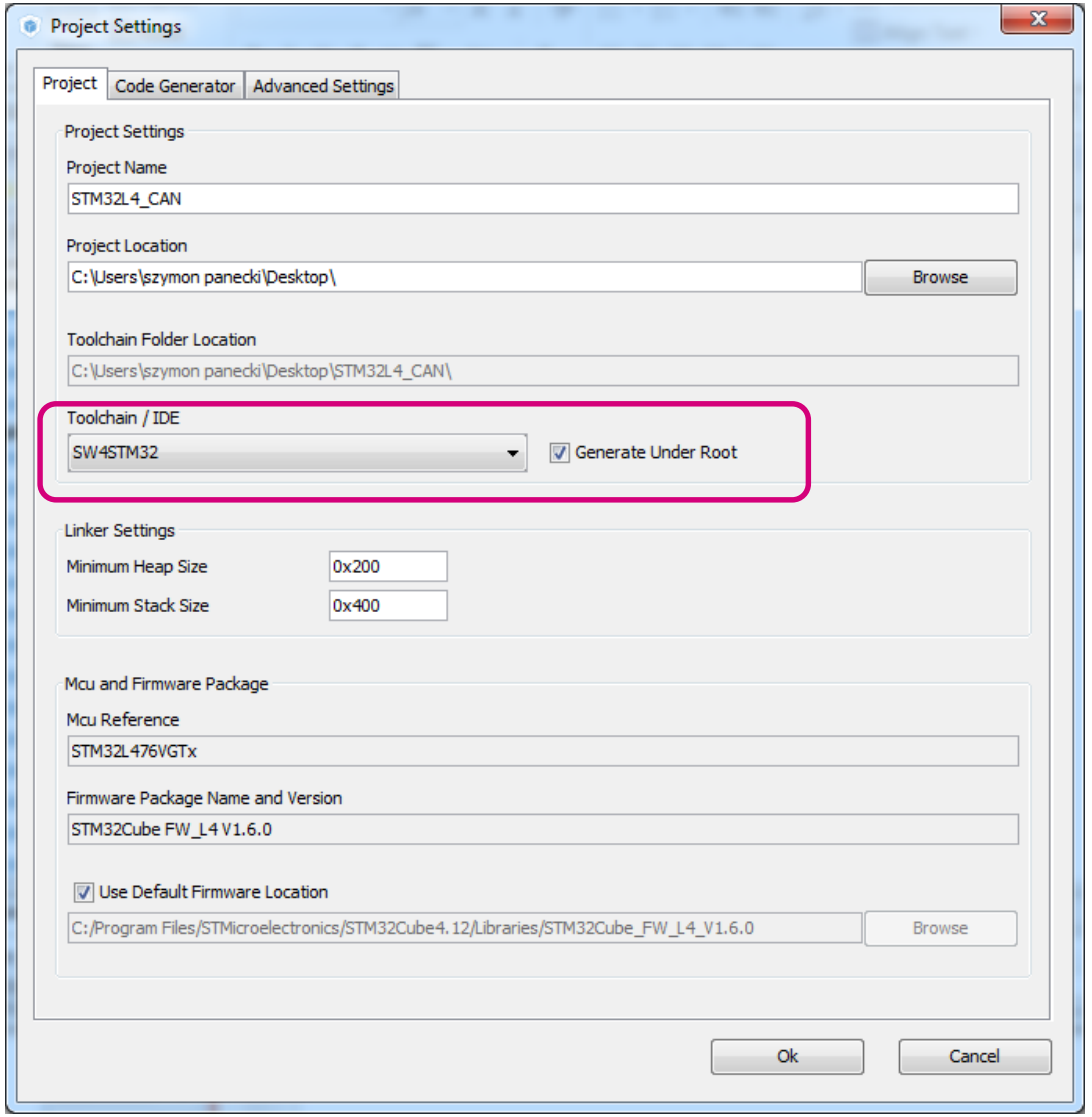


life.augmented

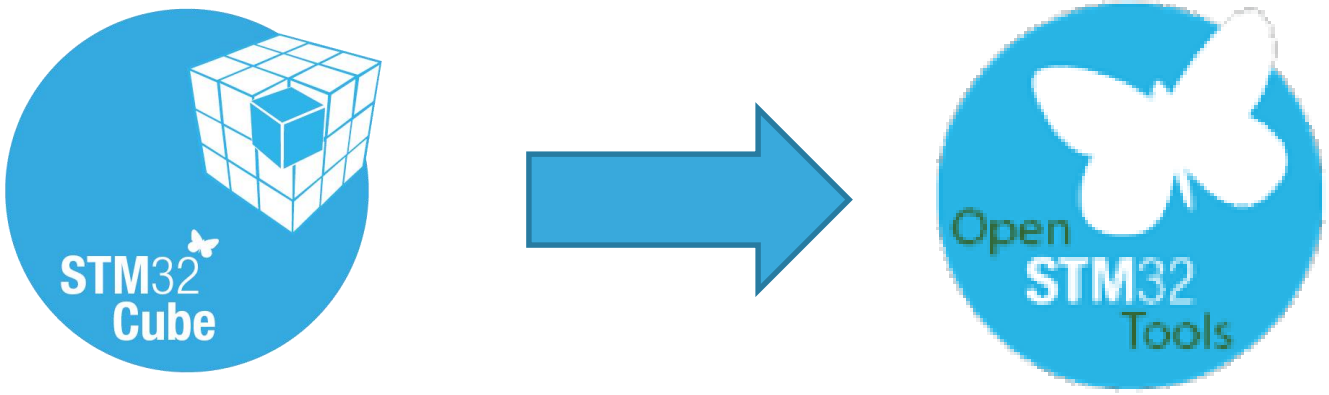
STM32CubeMX

Project generation

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code



life.augmented



- After successful code generation by STM32CubeMX this is the right time to import it into SW4STM32 toolchain for further processing

Modifying the code

data declaration - main.c file

Tasks:

1. Create structures for managing CAN (filters, tranmission message, reception message)
2. Configure filters in the way, that all received messages are accepted

```
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
CAN_FilterConfTypeDef  sFilterConfig;
CanTxMsgTypeDef        TxMessage;
CanRxMsgTypeDef        RxMessage;

/* USER CODE END PV */
```

CAN filter structure

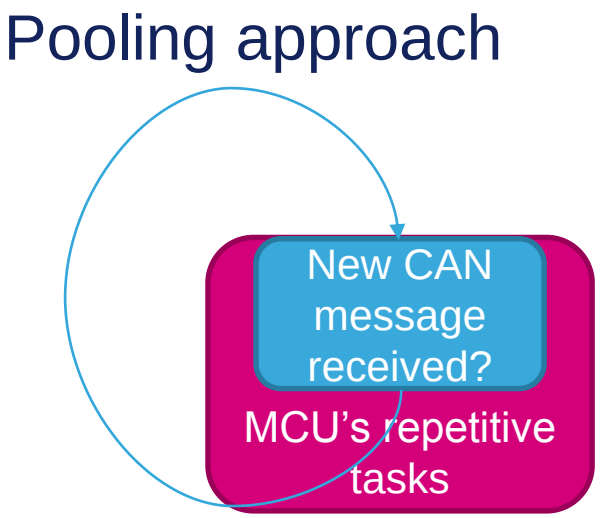
CAN transmission and reception structuresstructure

```
/* USER CODE BEGIN 2 */
 sFilterConfig.FilterNumber = 0;
 sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
 sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
 sFilterConfig.FilterIdHigh = 0x0000;
 sFilterConfig.FilterIdLow = 0x0000;
 sFilterConfig.FilterMaskIdHigh = 0x0000;
 sFilterConfig.FilterMaskIdLow = 0x0000;
 sFilterConfig.FilterFIFOAssignment = 0;
 sFilterConfig.FilterActivation = ENABLE;
 sFilterConfig.BankNumber = 0;

 HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig);

/* USER CODE END 2 */
```

CAN filter strucutre items configuration

CAN filter configuration function call

- Implementation of CAN message reception by pooling



Modifying the code

message transmission and reception - main.c file

Tasks:
1. Fill in structure for CAN message transmission
2. In infinite loop call two functions: to send and receive CAN message

```
/* USER CODE BEGIN 2 */
 TxMessage.StdId = 0x123;
 TxMessage.RTR = CAN_RTR_DATA;
 TxMessage.IDE = CAN_ID_STD;
 TxMessage.DLC = 8;
 TxMessage.Data[0] = 0x09;
 TxMessage.Data[1] = 0x10;
 TxMessage.Data[2] = 0x2A;
 TxMessage.Data[3] = 0x3B;
 TxMessage.Data[4] = 0x4C;
 TxMessage.Data[5] = 0x5D;
 TxMessage.Data[6] = 0x6E;
 TxMessage.Data[7] = 0x7F;

 /* USER CODE END 2 */
```

CAN message strucutre items configuration

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

 /* USER CODE BEGIN 3 */
 TxMessage.Data[0]++;

 HAL_CAN_Transmit(&hcan1,10);
 HAL_CAN_Receive(&hcan1, CAN_FIFO0, 100);

}
/* USER CODE END 3 */
```

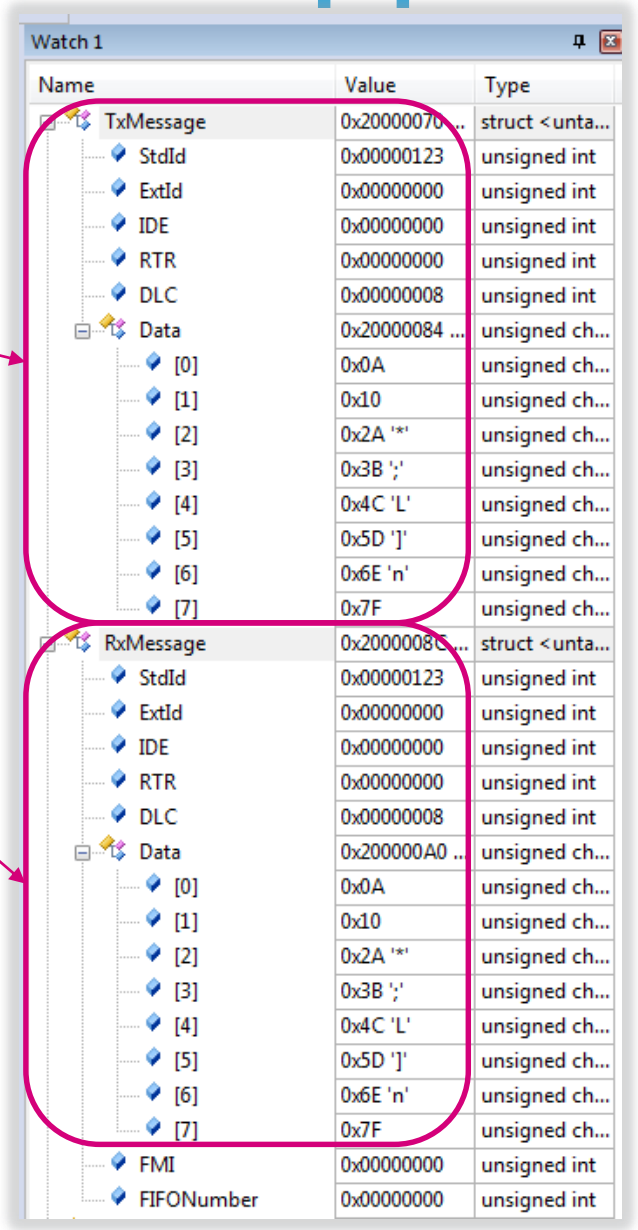
Incrementation of CAN message strucutre's data item with each loop iteration

Call of functions: to send and receive CAN message

life.augmented

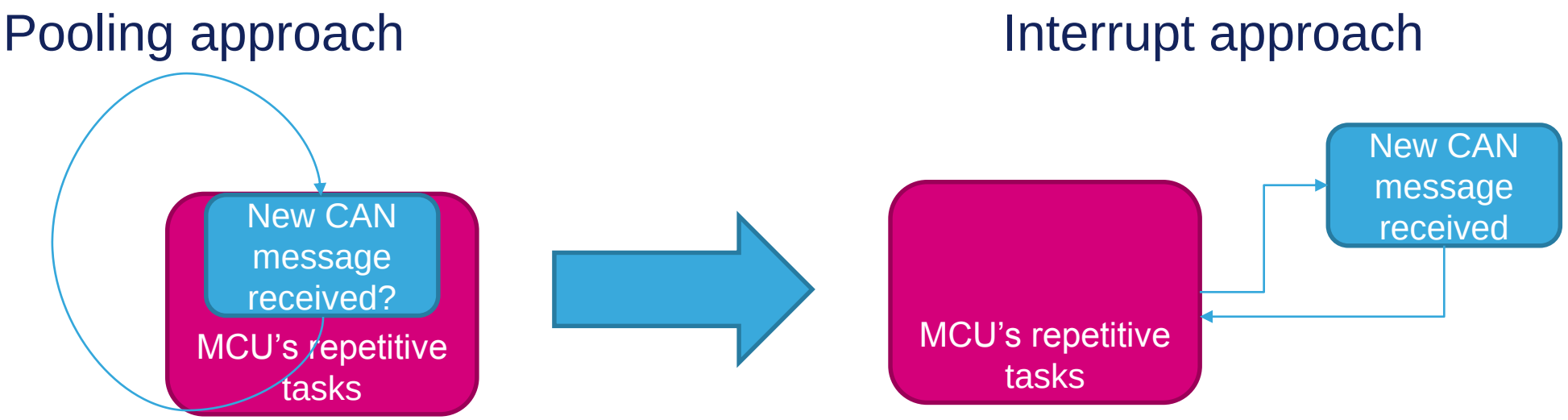
Running the application

- In debug session observe content of the
  - **TxMessage** transmission buffer
  - **RxMessage** reception buffer



life.augmented

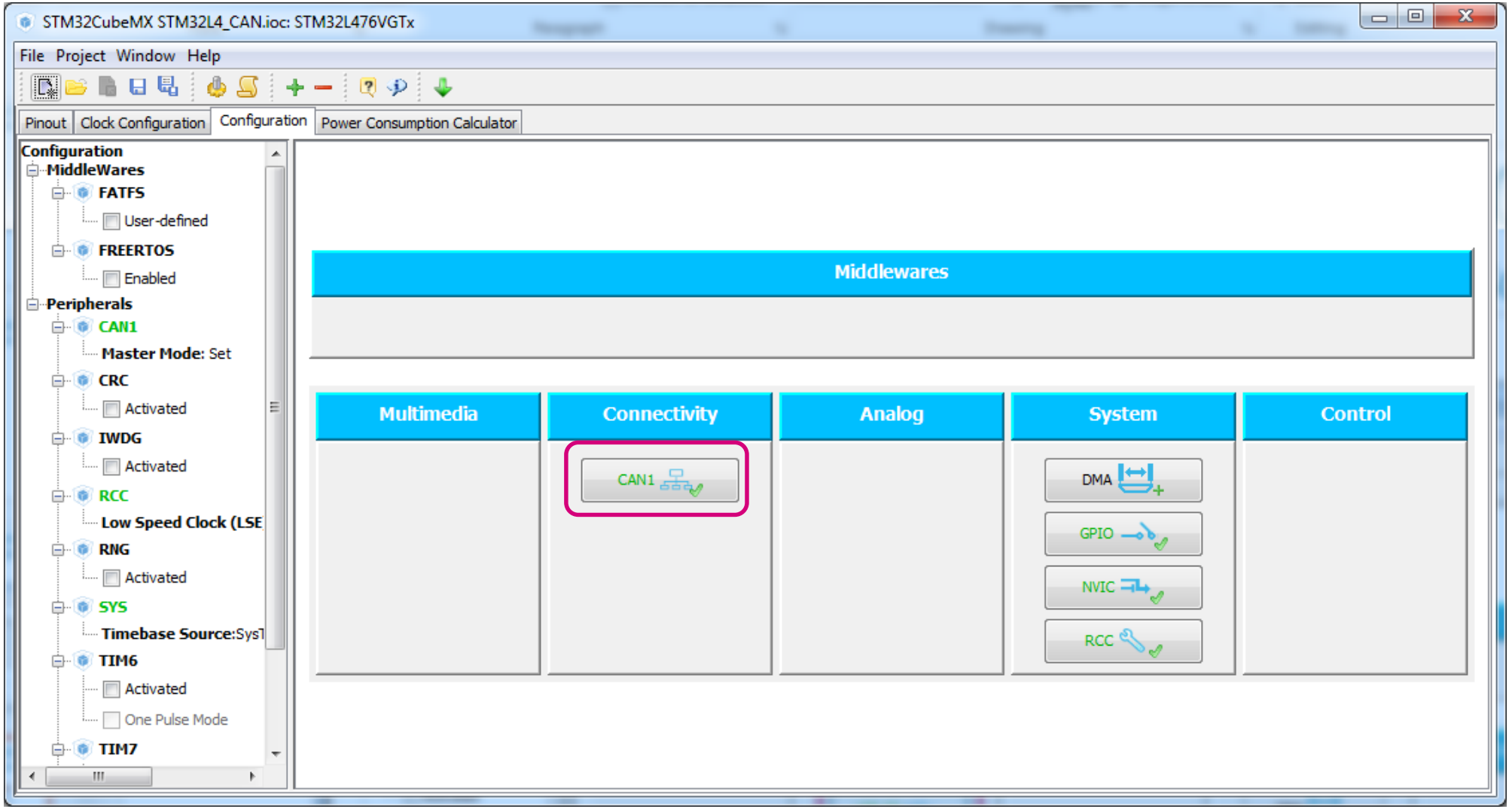
- Modification of application in order to replace CAN message reception by pooling with CAN message reception with interrupt



STM32CubeMX

Configure CAN

- Go to Configuration tab and select CAN peripheral

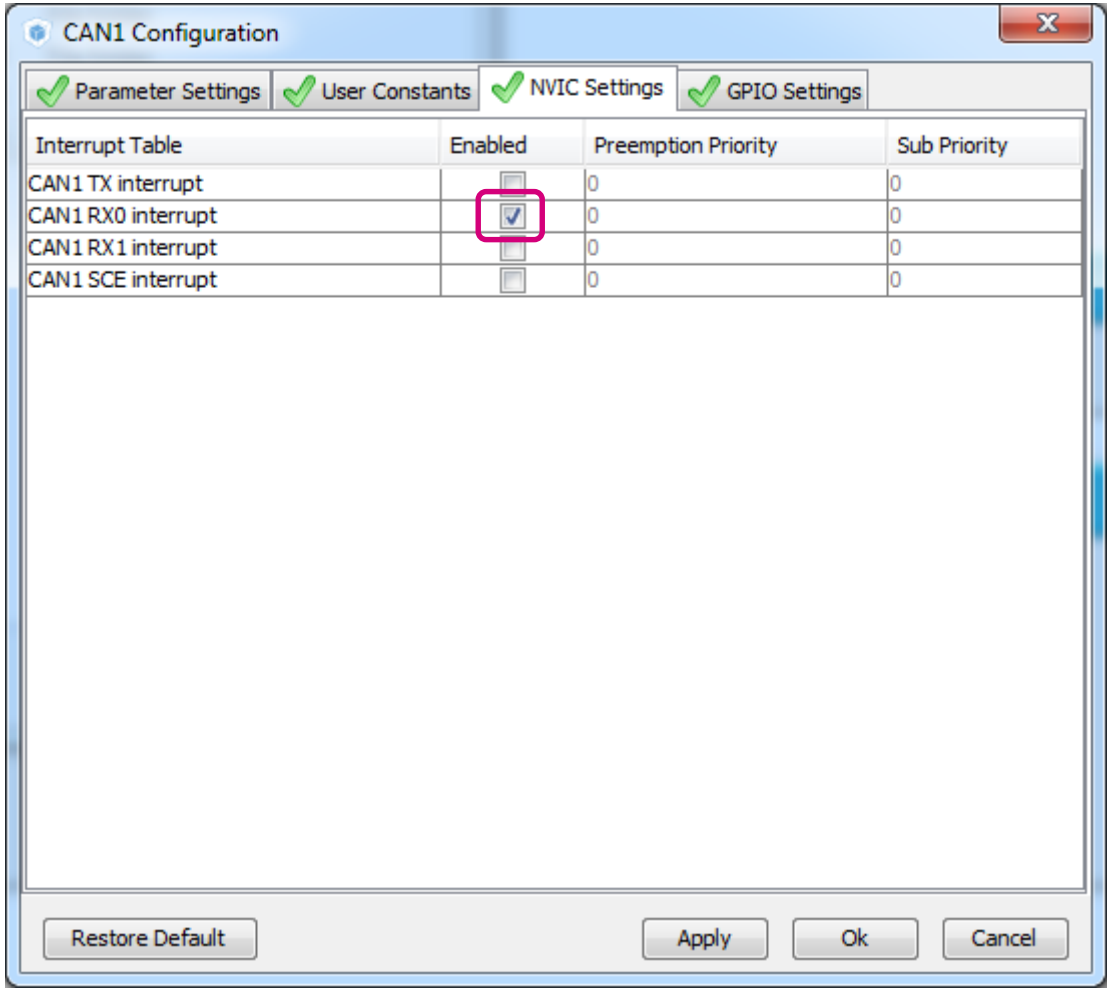


life.augmented

STM32CubeMX

enabling of CAN receive interrupt

- Select NVIC Settings tab
  - Enable CAN1RX0 interrupt
- Press **Ok** to confirm the configuration

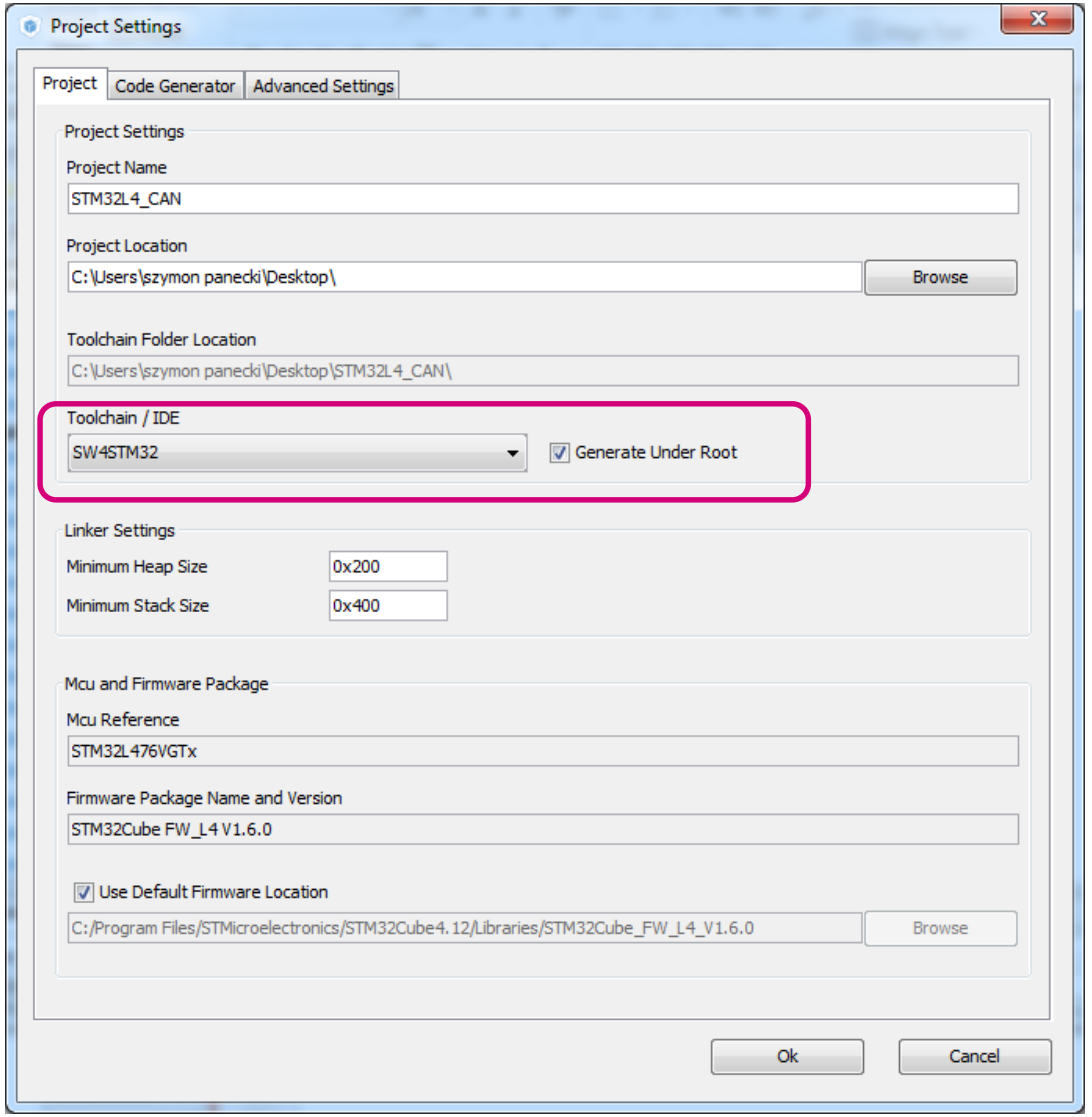


life.augmented

STM32CubeMX

Project generation

- Now we set the project details for generation
  - Menu > Project > Project Settings
  - Set the project name
  - Project location
  - Type of toolchain
- Now we can Generate Code
  - Menu > Project > Generate Code



life.augmented

Modifying the code

message transmission and reception - main.c file

Tasks:
1. Call a function to enable CAN reception interrupt
2. Remove call of function, which receives CAN messages in infinie while loop

```
/* USER CODE BEGIN 2 */
HAL_CAN_Receive_IT(&hcan1, CAN_FIFO0);

/* USER CODE END 2 */
```

Function call before while(1) loop to enable CAN reception interrupt

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

 /* USER CODE BEGIN 3 */
 TxMessage.Data[0]++;

 HAL_CAN_Transmit(&hcan1,10);
 HAL_CAN_Receive(&hcan1, CAN_FIFO0, 100);

}
/* USER CODE END 3 */
```

Incrementation of CAN message strucutre's data item with each loop iteration

Call of function to send CAN message

Comparing to pooling approach, call of this function should be removed, as CAN message reception will now be implemented in interrupt routine

life.augmented

Modifying the code

message reception - stm32l4xx_it.c file

Tasks:

1. In CAN reception interrupt handler call function to receive CAN message
2. In CAN reception interrupt handler call function to UNLOCK HAL after each interrupt generation

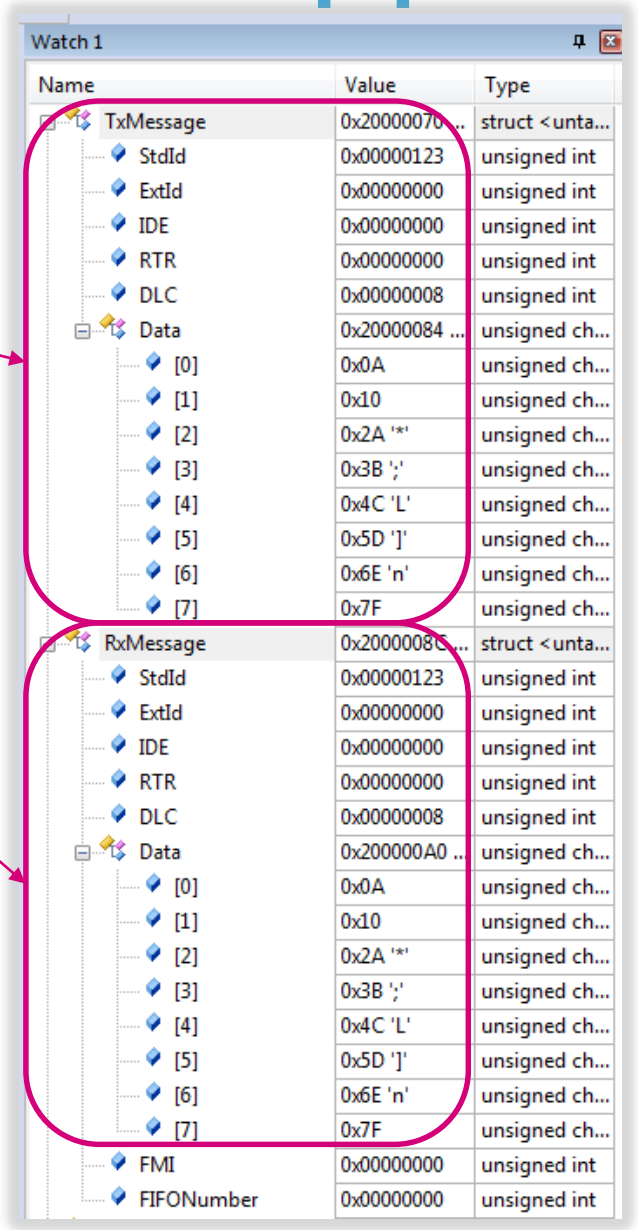
```
void CAN1_RX0_IRQHandler(void)
{
  /* USER CODE BEGIN CAN1_RX0_IRQn 0 */

  /* USER CODE END CAN1_RX0_IRQn 0 */
  HAL_CAN_IRQHandler(&hcan1);
  /* USER CODE BEGIN CAN1_RX0_IRQn 1 */

  __HAL_UNLOCK(&hcan1);

  HAL_CAN_Receive_IT(&hcan1, CAN_FIFO0);

  /* USER CODE END CAN1_RX0_IRQn 1 */
}
```

Call this function to release manually HAL for CAN structure

Call of function to receive CAN message in interrupt

Running the application

- In debug session observe content of the
  - **TxMessage** transmission buffer
  - **RxMessage** reception buffer



life.augmented

Enjoy!

www.st.com/mcu

life.augmented