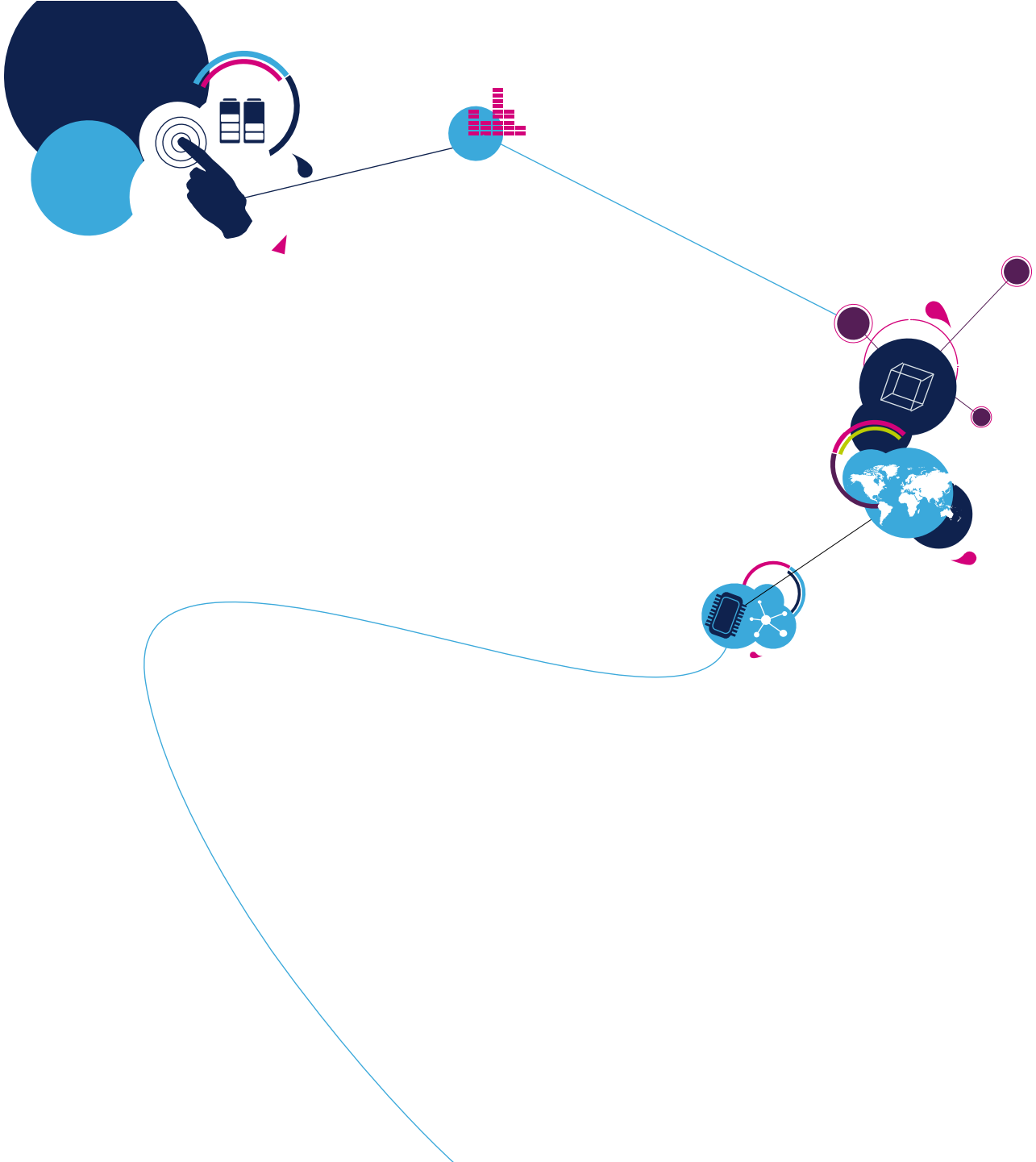
STM32L4 + CAN

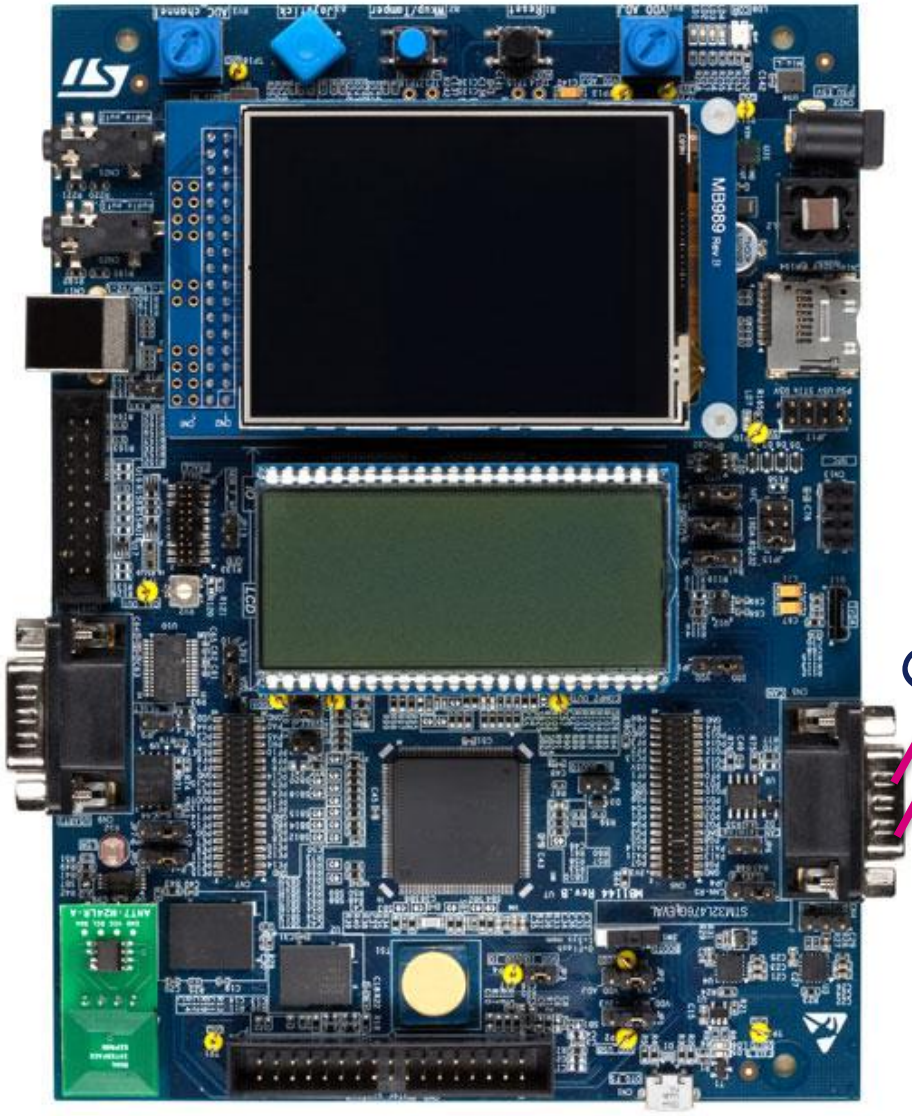


life.augmented

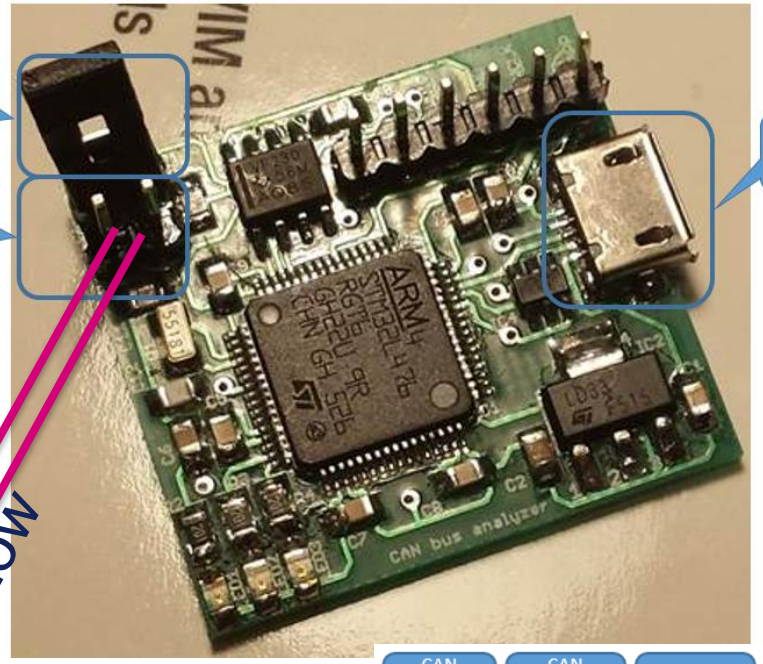
CAN connectivity

- Objective
  - Learn how to configure CAN in CubeMX and generate IDE project with code
  - Understand how to use HAL functions from new CAN API
- Method
  - Develop an application, which sends CAN messages and receives CAN messages

Hardware setup

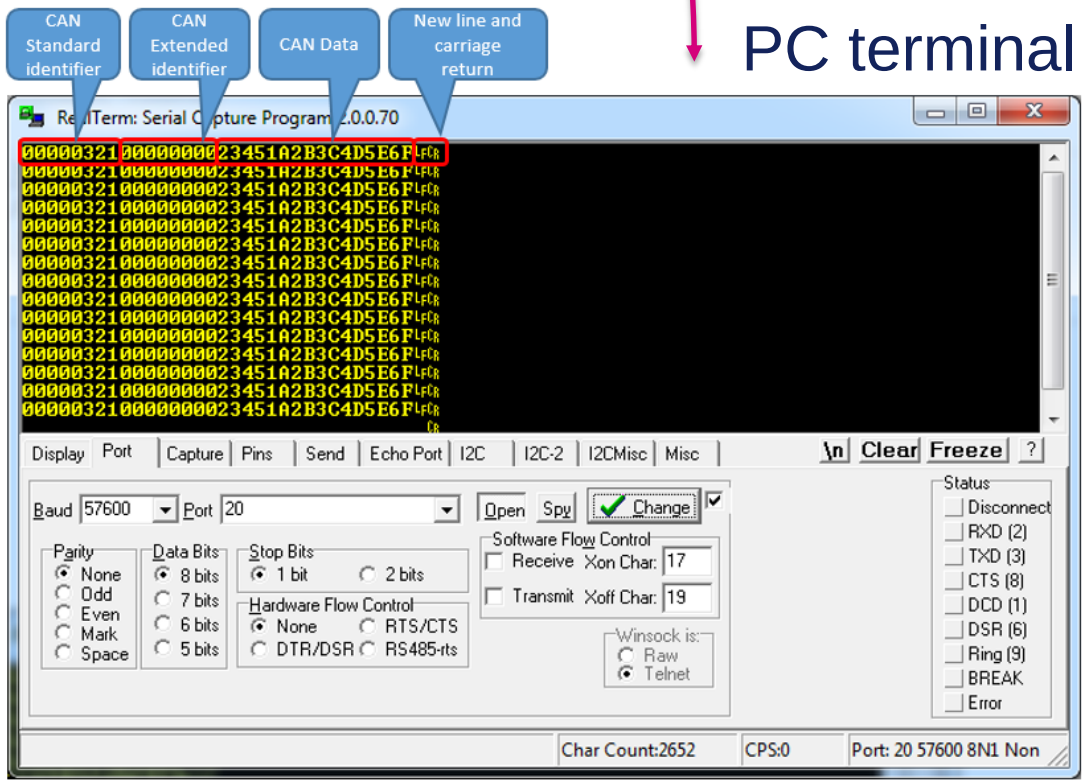


STM32L476G-EVAL



CAN bus analyzer

PC terminal

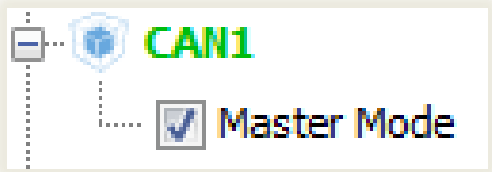


life.augmented

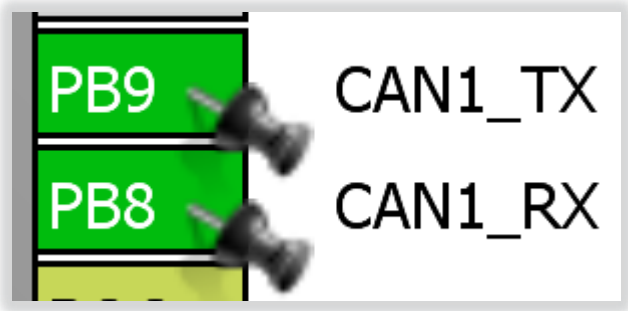
STM32CubeMX

Selecting CAN interface and clock

- Create project in STM32CubeMX
  - Menu > File > New Project
  - Select STM32L4 -> STM32L4x6 -> LQFP144 package -> STM32L476ZGTx
- Select CAN:
  - Select “**Master Mode**” for **CAN1**



  - Remap CAN pins to **PB8 and PB9**



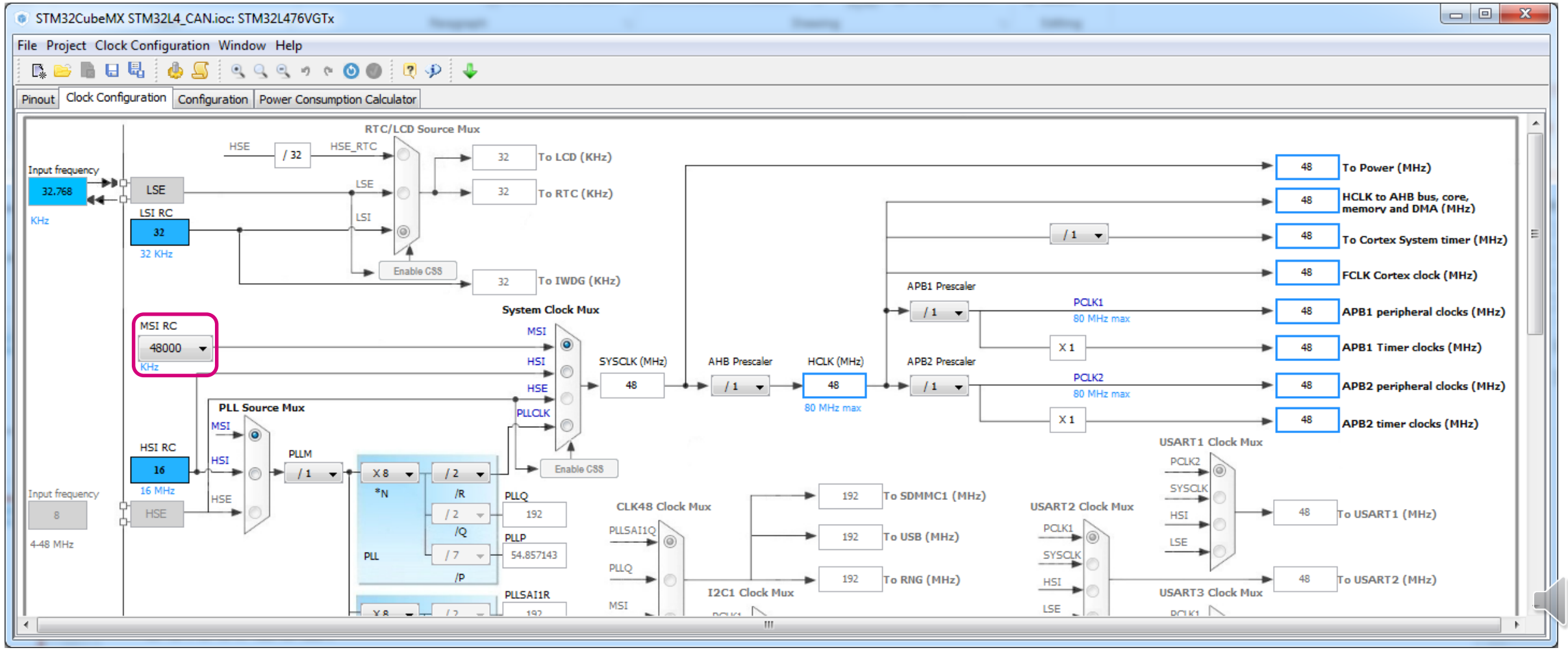
- Conigure GPIO to control LED

life.augmented

STM32CubeMX

clock configuration

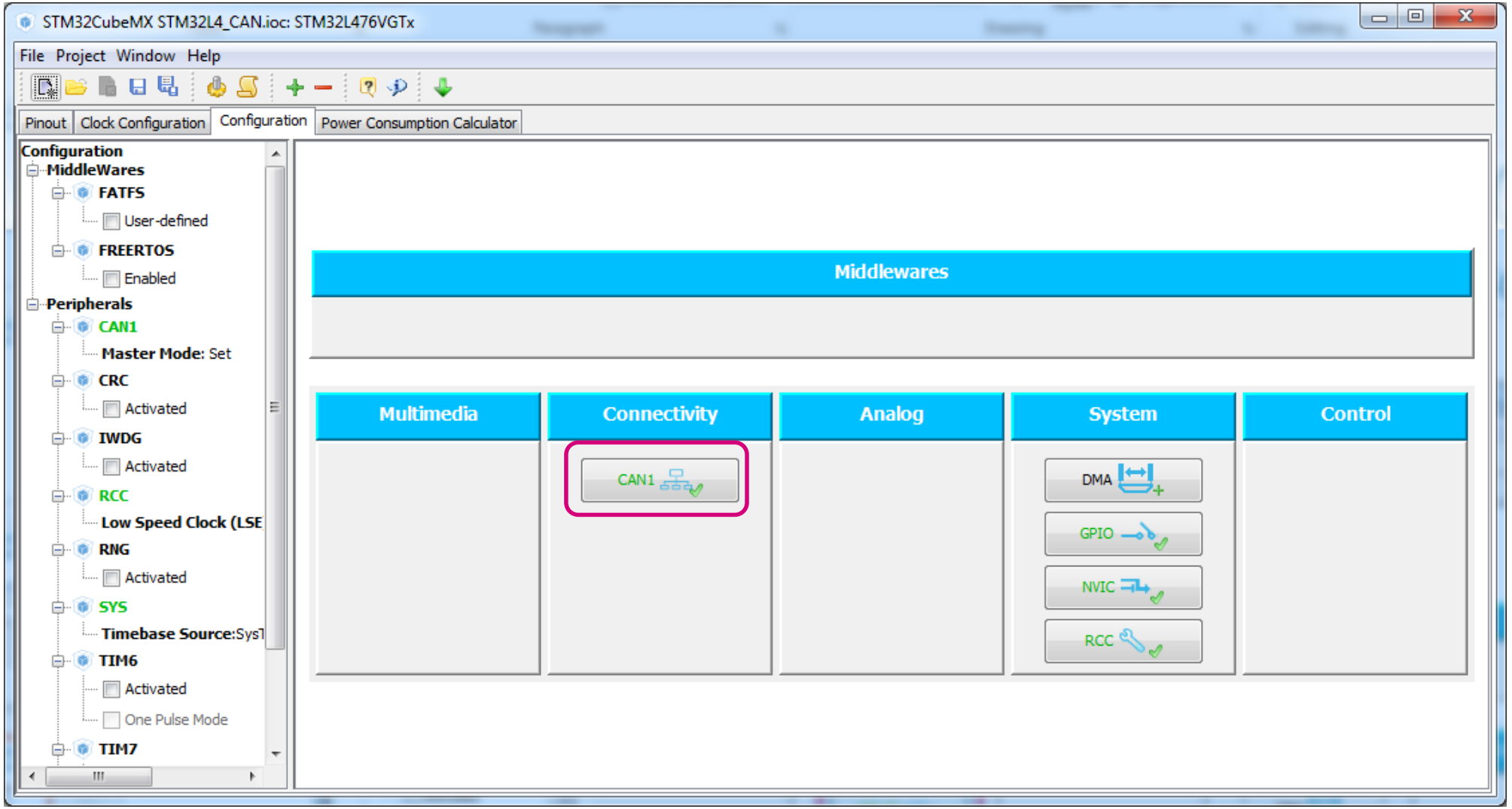
- Go to Clock Configuration tab and configure MCU clock system:
  - Change MSI default value (4 MHz) to 48 MHz



STM32CubeMX

Configure CAN

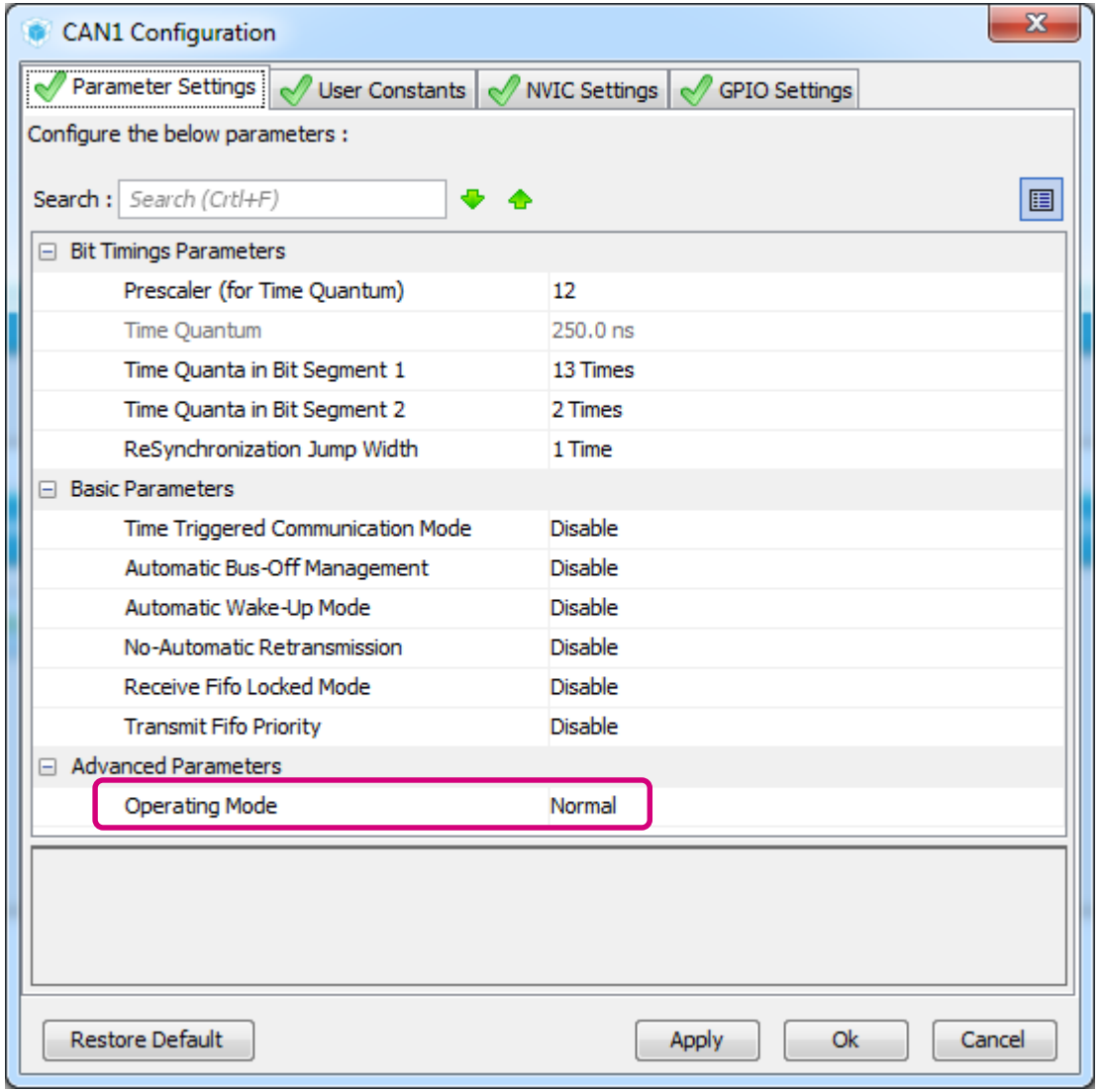
- Go to Configuration tab and select CAN peripheral



life.augmented

STM32CubeMX
configuration of CAN mode

- Select Parameter Settings tab
  - Change Operating Mode to Normal
- Press **Ok** to confirm the configuration

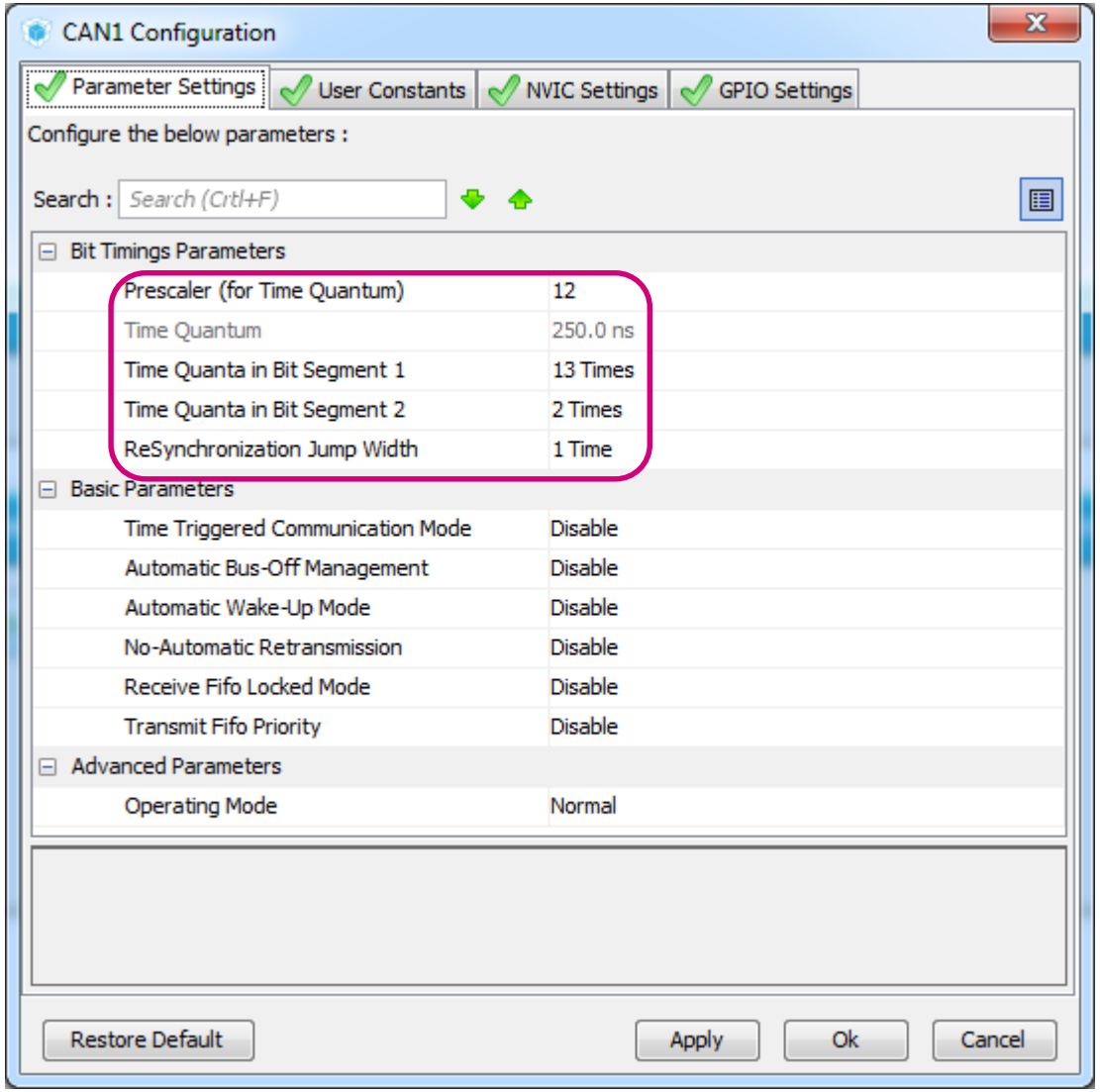


life.augmented

STM32CubeMX

configuration of CAN baudrate

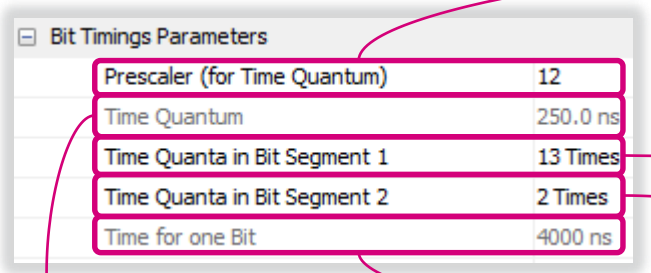
- Select Parameter Settings tab
  - Fill in Bit Timing Parameters to set CAN baudrate
- Press **Ok** to confirm the configuration

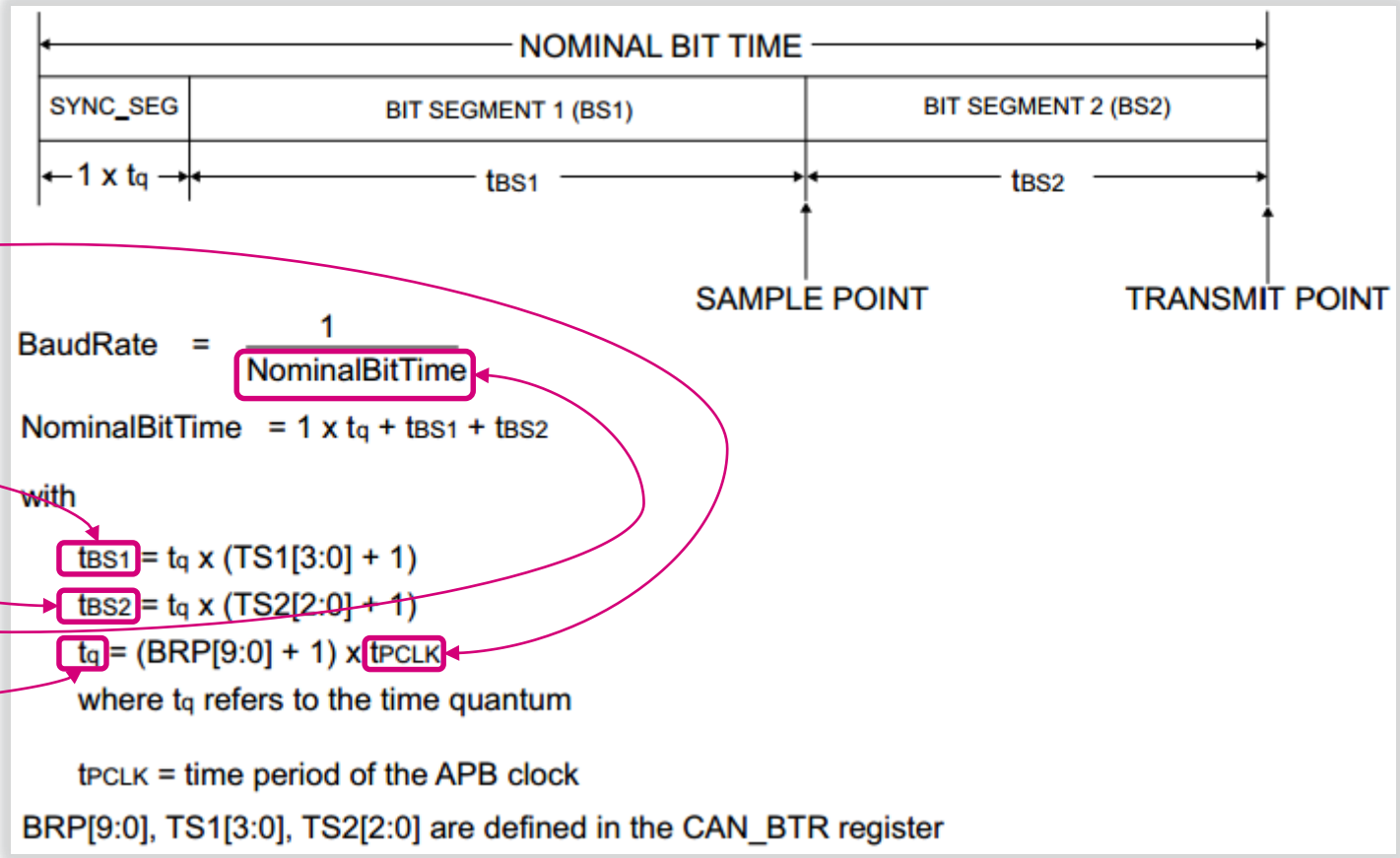


life.augmented

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

- Formula used to calculate CAN baudrate





- More information:

life.augmented

RM0351
Reference manual

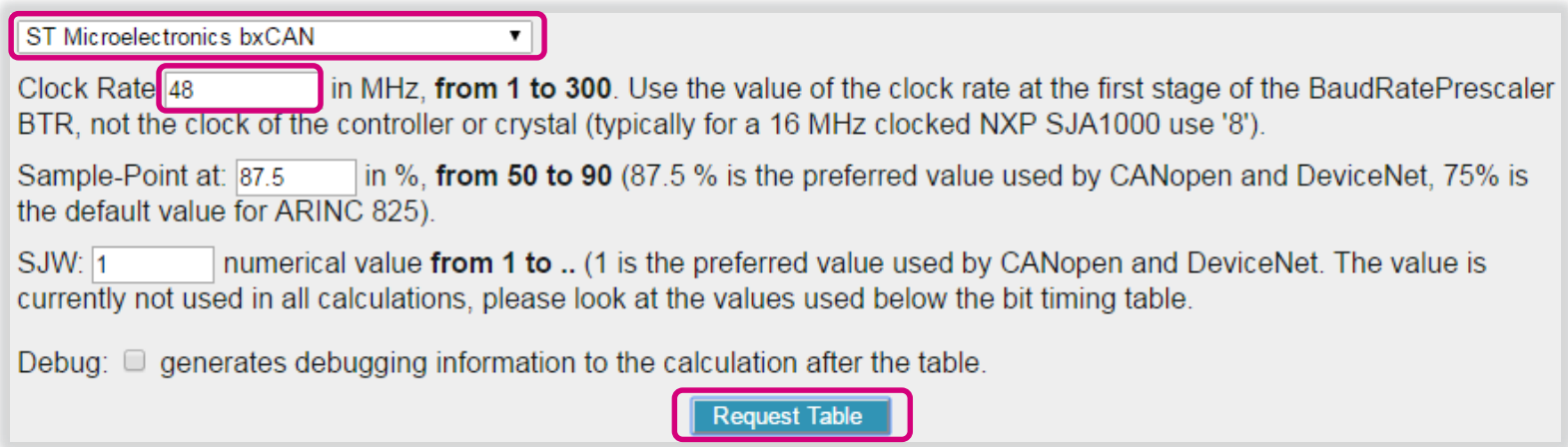
STM32L4x5 and STM32L4x6 advanced ARM®-based 32-bit MCUs

46 Controller area network (bxCAN) 1576

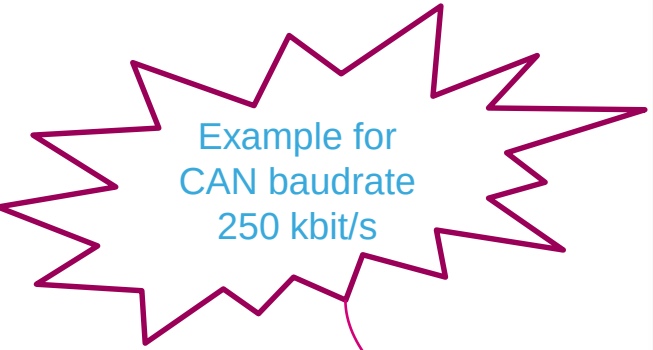
Easy configuration of CAN baudrate

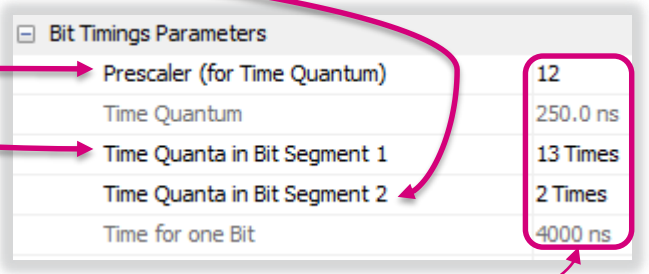
- http://www.bittiming.can-wiki.info/ webpage allows to obtain CAN baudrate configuation parameters automatically

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



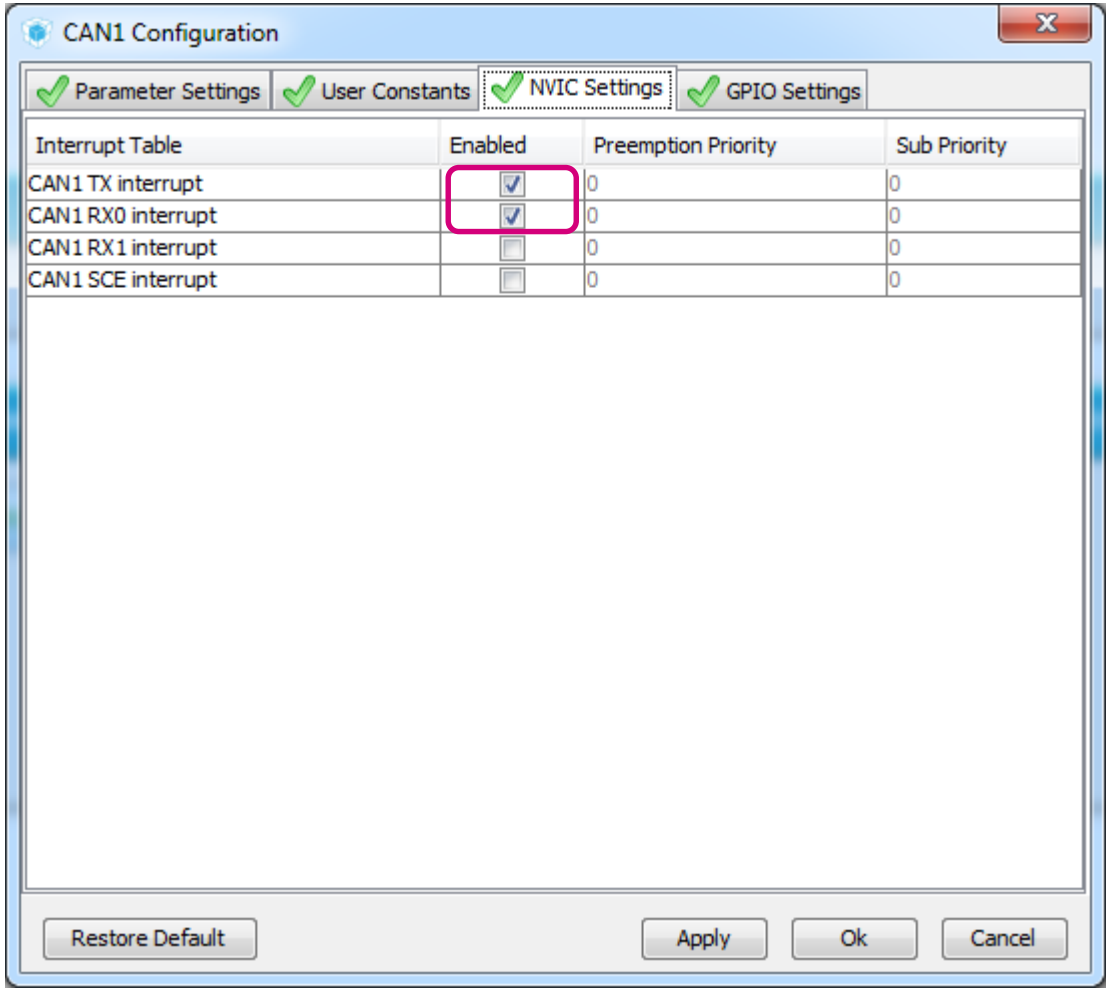


life.augmented

STM32CubeMX

configuration of CAN interrupts

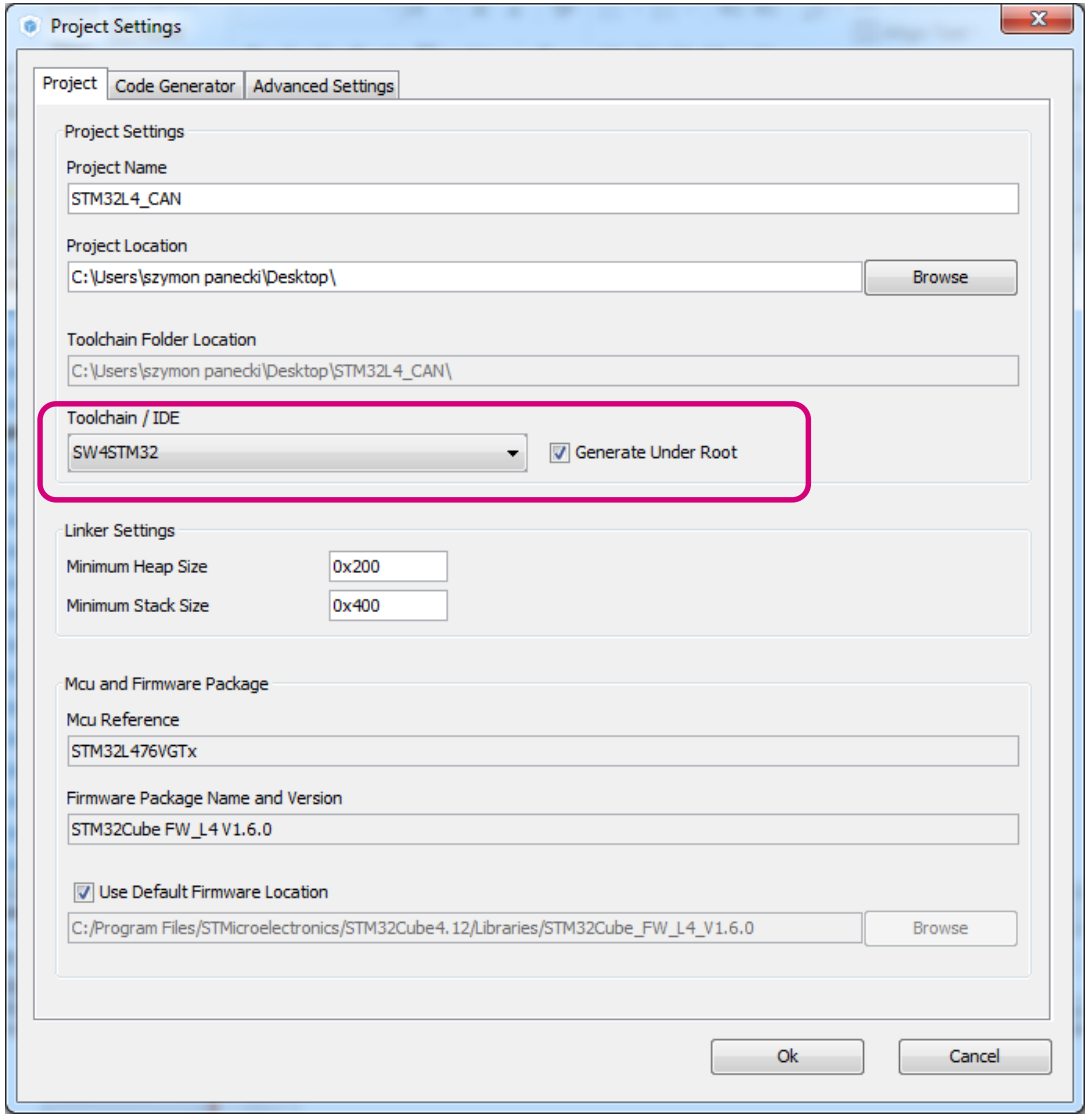
- Select NVIC Settings tab
  - Enable CAN1 TX interrupt
  - Enable CAN1 RX0 interrupt
- Press **Ok** to confirm the configuration



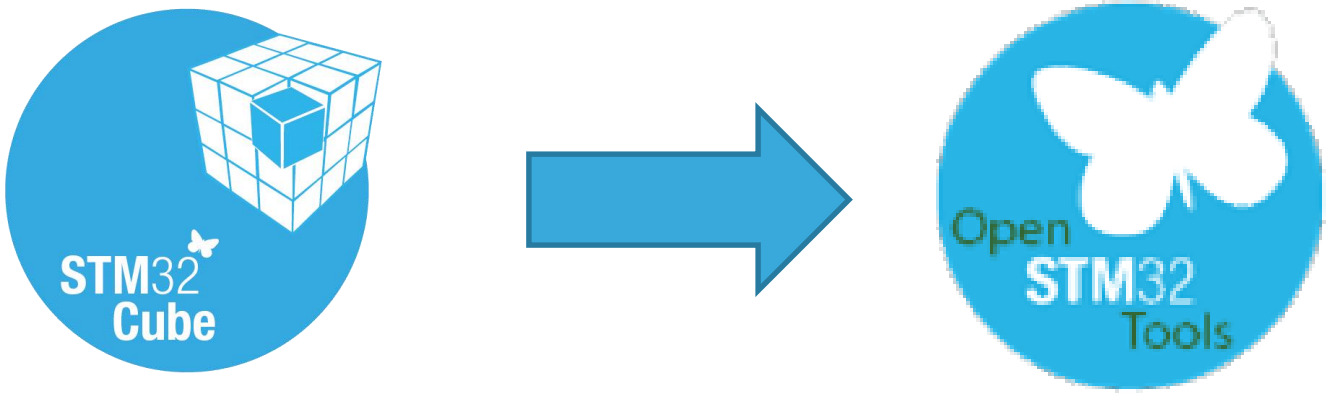
STM32CubeMX

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 code development

Modifying the code

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

```
/* USER CODE BEGIN 2 */
  sFilterConfig.FilterBank = 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 = CAN_RX_FIFO0;
  sFilterConfig.FilterActivation = ENABLE;
  sFilterConfig.SlaveStartFilterBank = 14;

  if (HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig) != HAL_OK)
  {
    /* Filter configuration Error */
    Error_Handler();
  }
```

CAN filter strucutre items configuration

CAN filter configuration function call

life.augmented

Modifying the code

main.c file

Tasks:
1. Start CAN
2. Enable notifications
3. Fill in TX message

```
if (HAL_CAN_Start(&hcan1) != HAL_OK)
 {
   /* Start Error */
   Error_Handler();
 }
```

CAN start

```
if (HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_TX_MAILBOX_EMPTY) != HAL_OK)
 {
   /* Notification Error */
   Error_Handler();
 }
```

CAN notifications (interrupts)

```
  TxHeader.StdId = 0x321;
  TxHeader.ExtId = 0x01;
  TxHeader.RTR = CAN_RTR_DATA;
  TxHeader.IDE = CAN_ID_STD;
  TxHeader.DLC = 8;
  TxHeader.TransmitGlobalTime = DISABLE;
  TxData[0] = 1;
  TxData[1] = 2;
  TxData[2] = 3;
  TxData[3] = 4;
  TxData[4] = 5;
  TxData[5] = 6;
  TxData[6] = 7;
  TxData[7] = 8;

  /* USER CODE END 2 */
```

CAN TX message

Modifying the code

main.c file

Tasks:
1. Send message in infinite loop

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

 /* USER CODE END WHILE */

 /* USER CODE BEGIN 3 */
     HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
     HAL_Delay(500);
     TxData[7] = TxData[7] + 1;
 }
 /* USER CODE END 3 */
```

CAN send message

Modifying the code

main.c file

Tasks:
1. Interrupts callbacks

```
/* USER CODE BEGIN 4 */
void HAL_CAN_TxMailbox0CompleteCallback(CAN_HandleTypeDef *hcan)
{
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_2);
}

void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &RxHeader, RxData);
    HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_2);
}
/* USER CODE END 4 */
```

CAN TX callback

CAN RX callback

life.augmented

The same application using old CAN API

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

Modifying the code

message transmission and reception - main.c file

Tasks:

1. Fill in structure for CAN message transmission
2. Enable CAN RX interrupt
3. 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;

HAL_CAN_Receive_IT(&hcan1, CAN_FIFO0);


 /* USER CODE END 2 */
```

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

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

 HAL_CAN_Transmit(&hcan1,10);

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

CAN message strucutre items configuration

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

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

Call of function to send CAN message

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

Migration guide
Old CAN API -> New CAN API

Short migration guide

- Fields of **CAN_InitTypeDef** structure are renamed : **SJW** to **SyncJumpWidth**, **BS1** to **TimeSeg1**, **BS2** to **TimeSeg2**, **ABOM** to **AutoBusOff**, **AWUM** to **AutoWakeUp**, **NART** to **AutoRetransmission** (inversed), **RFLM** to **ReceiveFifoLocked** and **TXFP** to **TransmitFifoPriority**
- **HAL_CAN_Init()** is split into both **HAL_CAN_Init()** and **HAL_CAN_Start()**
- **HAL_CAN_Transmit()** is replaced by **HAL_CAN_AddTxMessage()** to place Tx request, then **HAL_CAN_GetTxMailboxesFreeLevel()** for polling until completion
- **HAL_CAN_Transmit_IT()** is replaced by **HAL_CAN_ActivateNotification()** to enable transmission with interrupt mode, then **HAL_CAN_AddTxMessage()** to place Tx request
- **HAL_CAN_Receive()** is replaced by **HAL_CAN_GetRxFifoFillLevel()** for polling until reception, then **HAL_CAN_GetRxMessage()** to get Rx message
- **HAL_CAN_Receive_IT()** is replaced by **HAL_CAN_ActivateNotification()** to enable reception with interrupt mode, then **HAL_CAN_GetRxMessage()** in the receive callback to get Rx message
- **HAL_CAN_Sleep()** is renamed to **HAL_CAN_RequestSleep()**
- **HAL_CAN_TxCpltCallback()** is split into **HAL_CAN_TxMailbox0CompleteCallback()**, **HAL_CAN_TxMailbox1CompleteCallback()** and **HAL_CAN_TxMailbox2CompleteCallback()**
- **HAL_CAN_RxCpltCallback()** is split into **HAL_CAN_RxFifo0MsgPendingCallback()** and **HAL_CAN_RxFifo1MsgPendingCallback()**

Short migration guide

- More complete "how to use the new driver" is detailed in the driver header section itself

- The legacy HAL CAN driver is also present in the release in Drivers/STM32L4xx_HAL_Driver/Src/Legacy and Drivers/STM32L4xx_HAL_Driver/Inc/Legacy folders for software compatibility reasons. Its usage is not recommended as deprecated i.e. no more maintenance will be done for the old driver. It can however be enabled through switch **HAL_CAN_LEGACY_MODULE_ENABLED** in **stm32l4xx_hal_conf.h**

Enjoy!

www.st.com/mcu

life.augmented