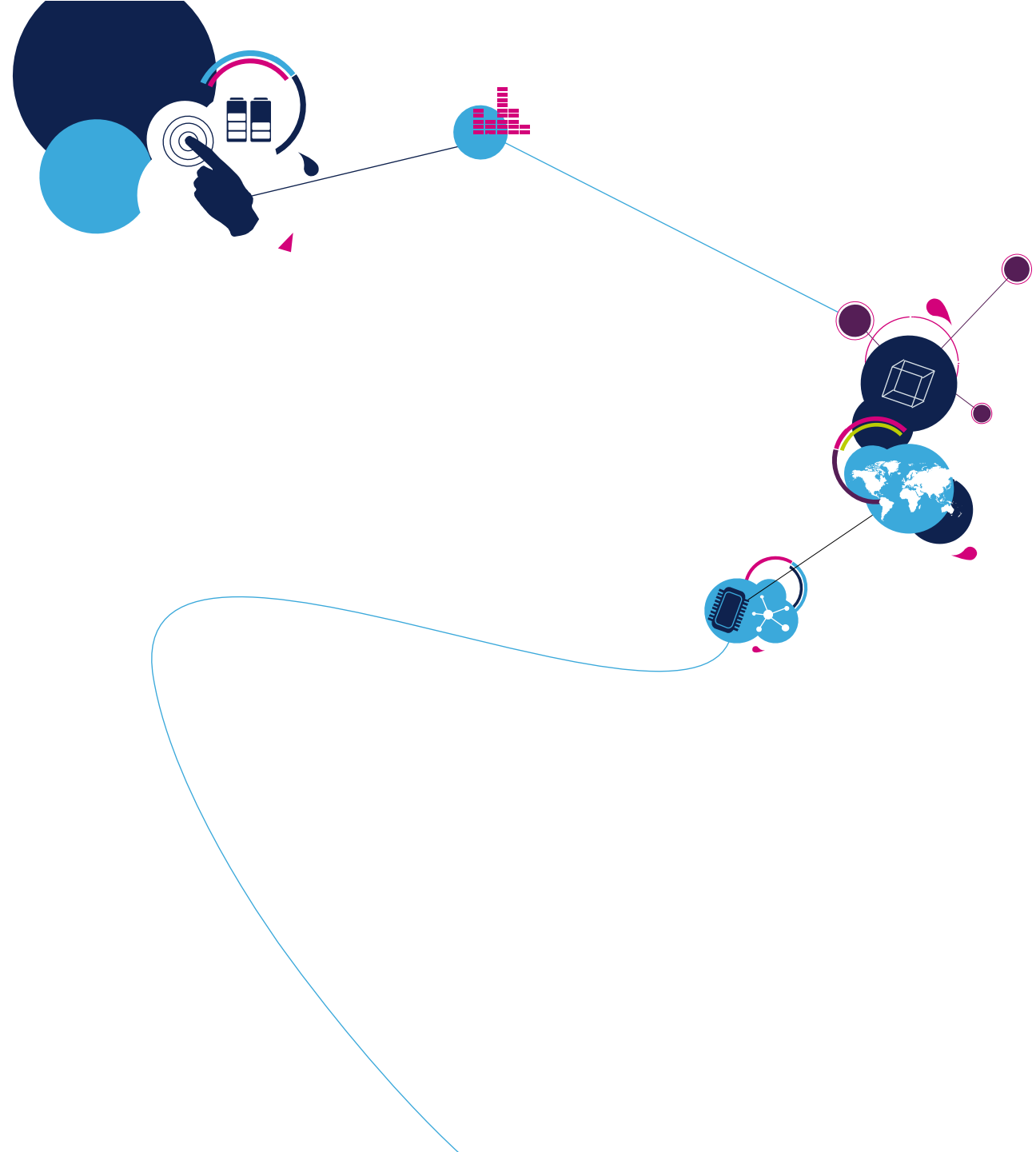


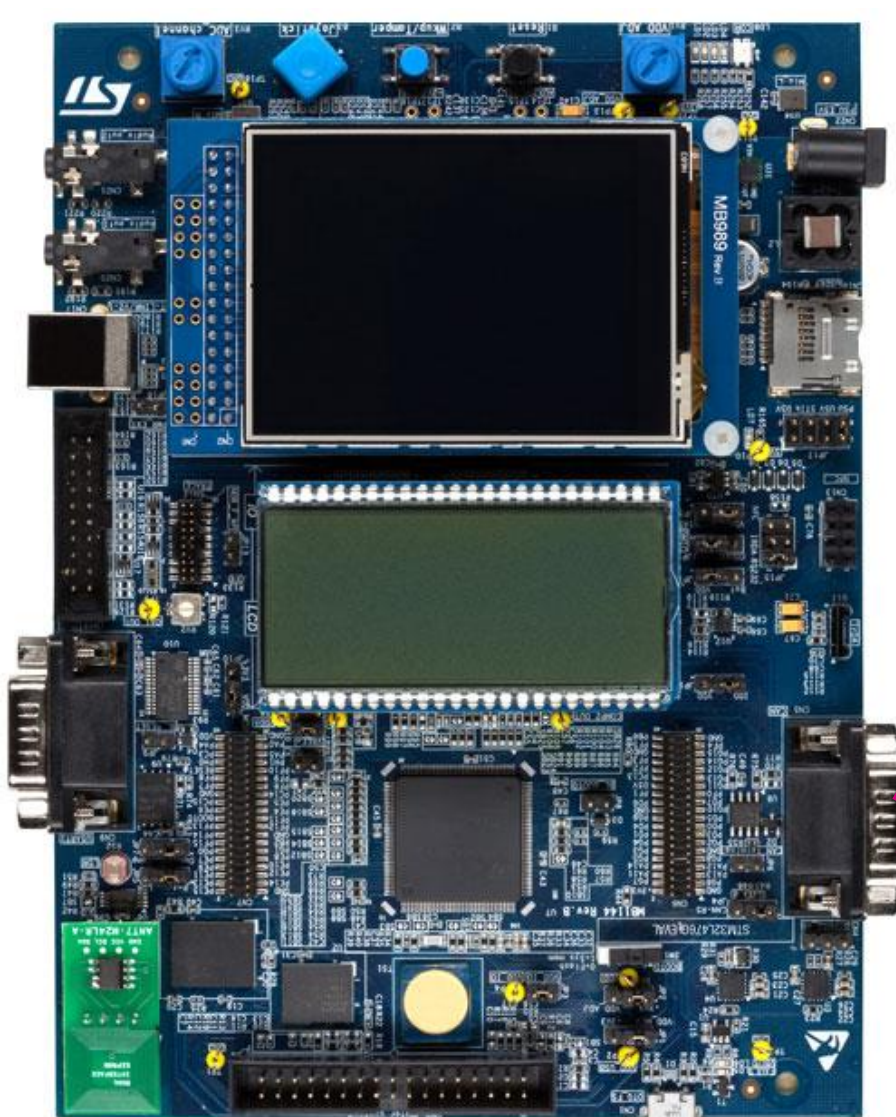
STM32L4 + CAN



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



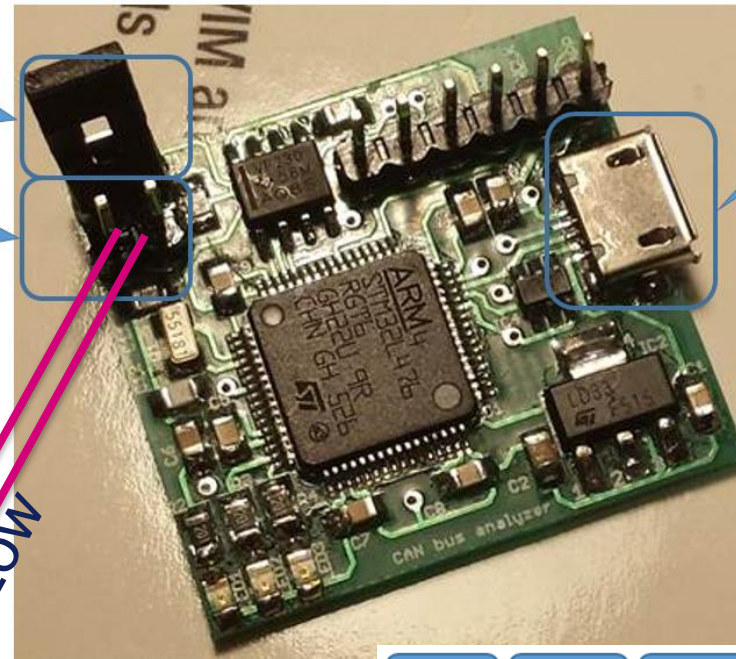
STM32L476G-EVAL

CAN High
CAN Low

CAN bus analyzer

CAN termination jumper

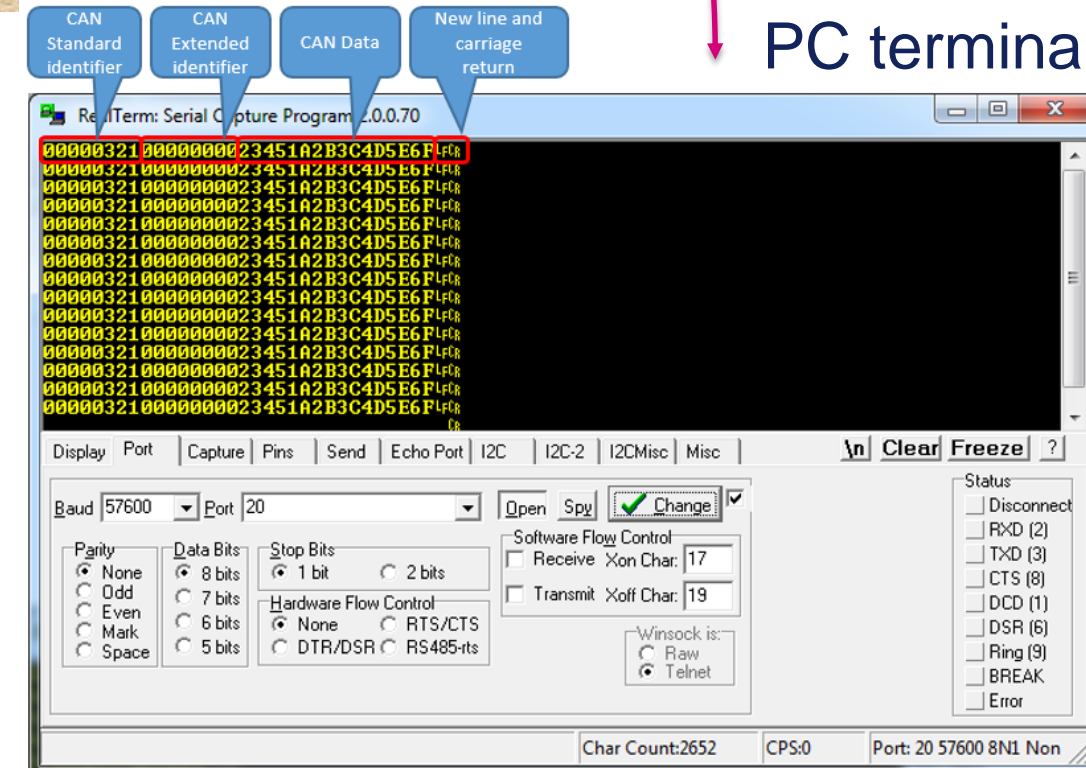
CAN High and CAN low pins



USB connector

USB VCP

PC terminal



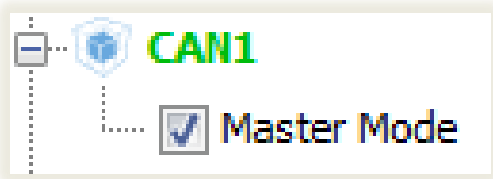
STM32CubeMX

Selecting CAN interface and clock

- Create project in STM32CubeMX
 - Menu > File > New Project
 - Select STM32L4 -> STM32L4x6 -> LQFP144 package -> STM32L476ZGTx

- Select CAN:

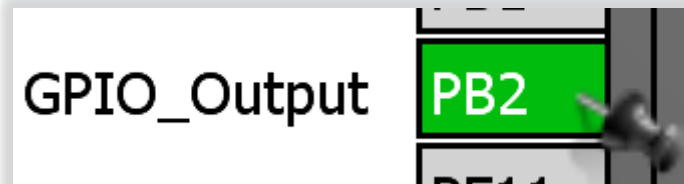
- Select “**Master Mode**” for **CAN1**



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



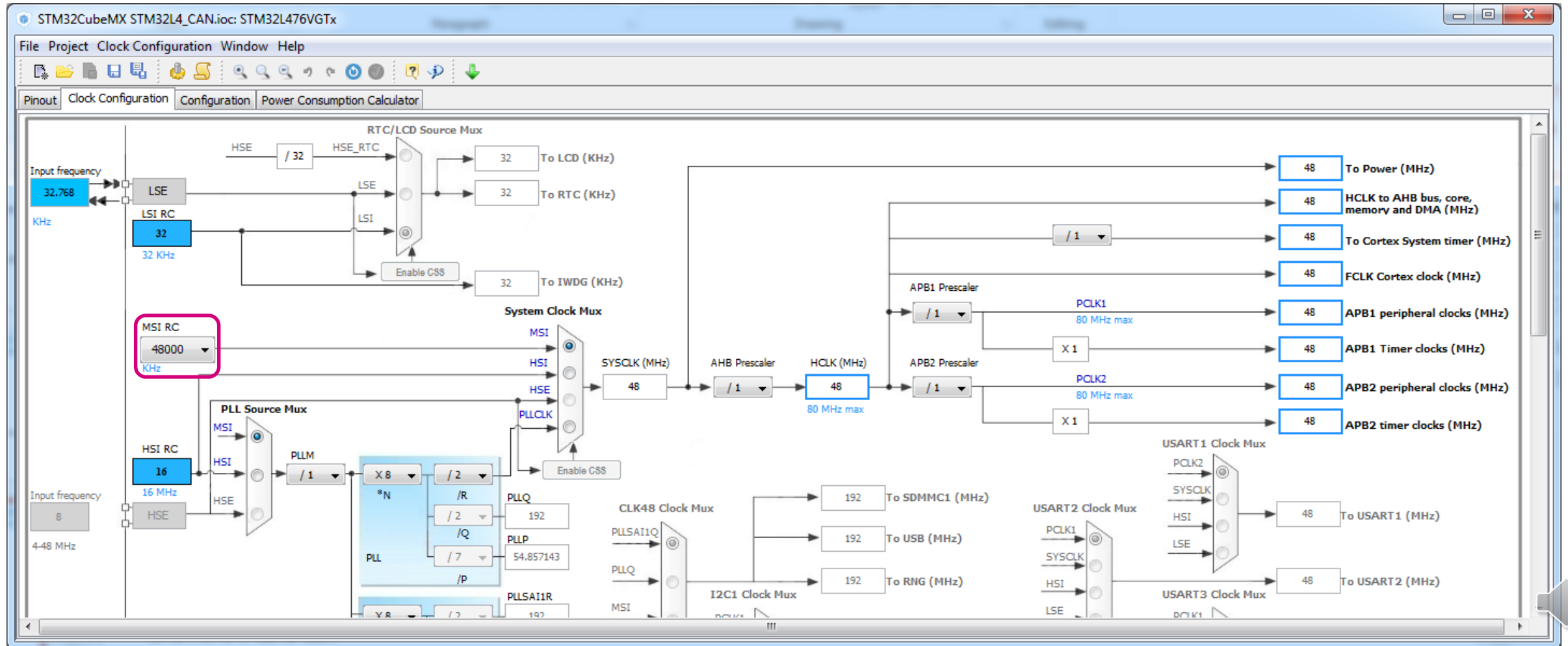
- Configure GPIO to control LED



STM32CubeMX

clock configuration

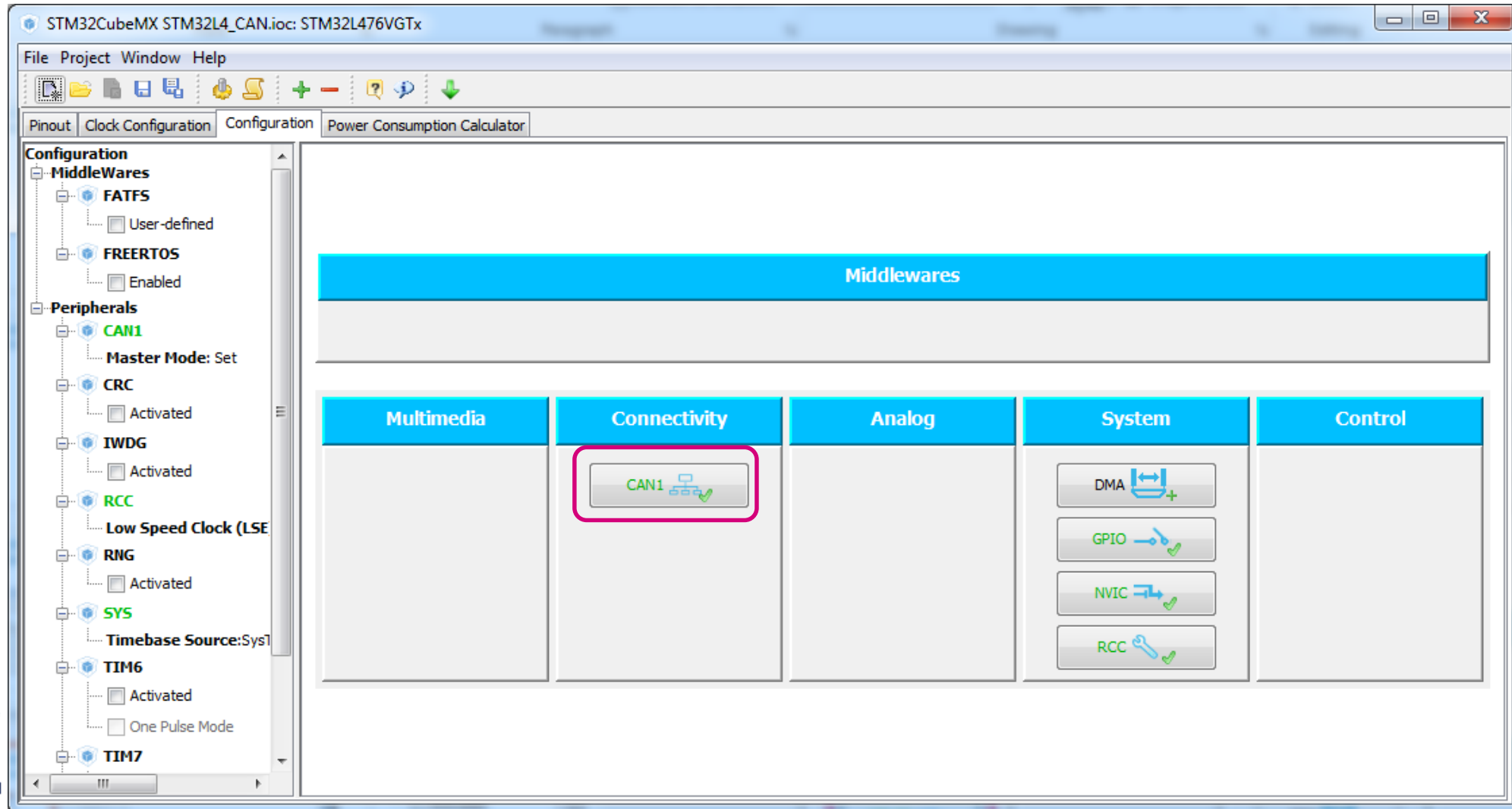
- Go to Clock Configuration tab and configure MCU clock system:
 - Change MSI default value (4 MHz) to 48 MHz



STM32CubeMX

Configure CAN

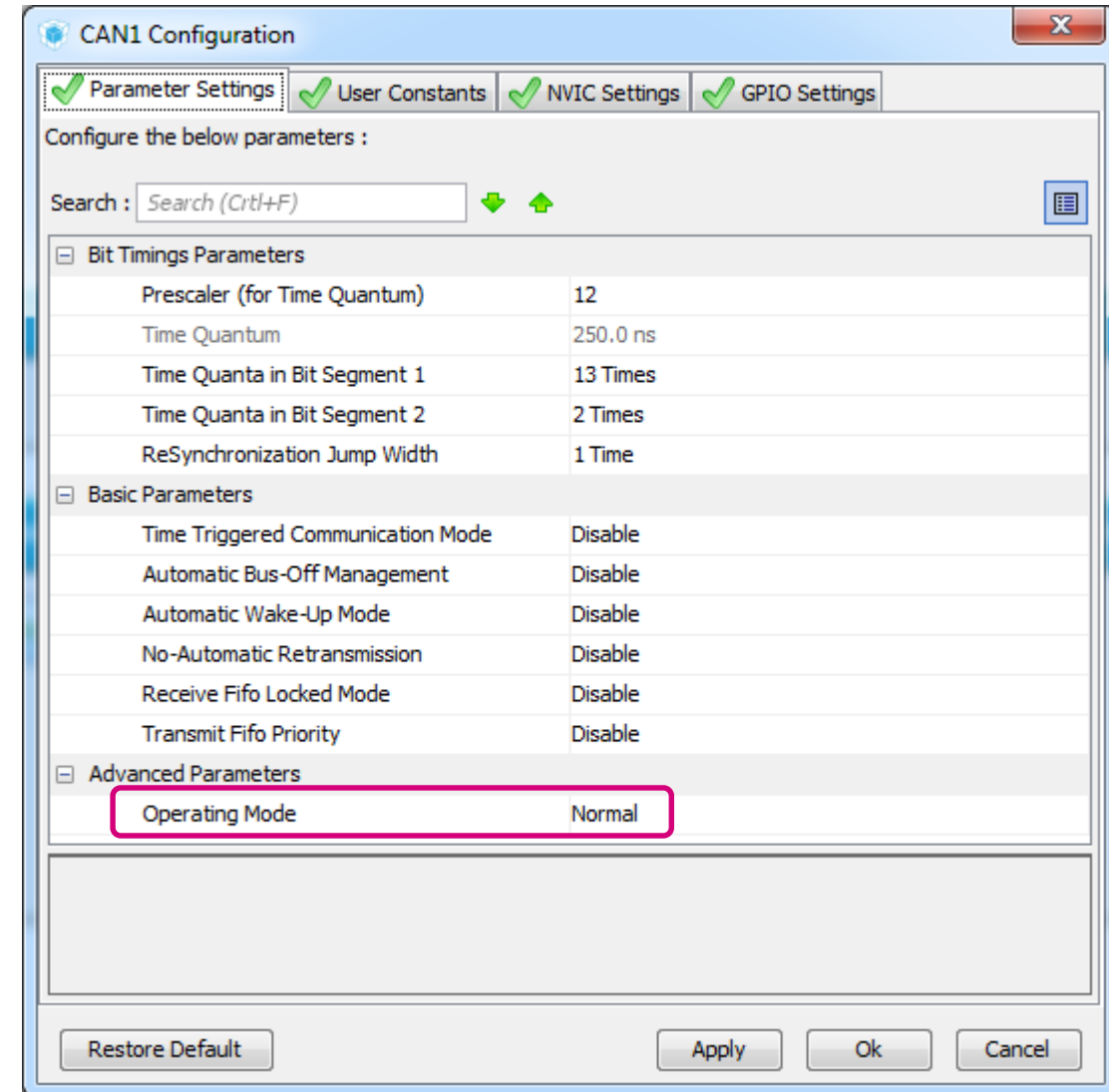
- Go to Configuration tab and select CAN peripheral



STM32CubeMX

configuration of CAN mode

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



STM32CubeMX

configuration of CAN baudrate

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

CAN1 Configuration

Parameter Settings User Constants NVIC Settings GPIO Settings

Configure the below parameters :

Search : Search (Ctrl+F)

☒ Bit Timings Parameters

Prescaler (for Time Quantum)	12
Time Quantum	250.0 ns
Time Quanta in Bit Segment 1	13 Times
Time Quanta in Bit Segment 2	2 Times
ReSynchronization Jump Width	1 Time

☒ Basic Parameters

Time Triggered Communication Mode	Disable
Automatic Bus-Off Management	Disable
Automatic Wake-Up Mode	Disable
No-Automatic Retransmission	Disable
Receive Fifo Locked Mode	Disable
Transmit Fifo Priority	Disable

☒ Advanced Parameters

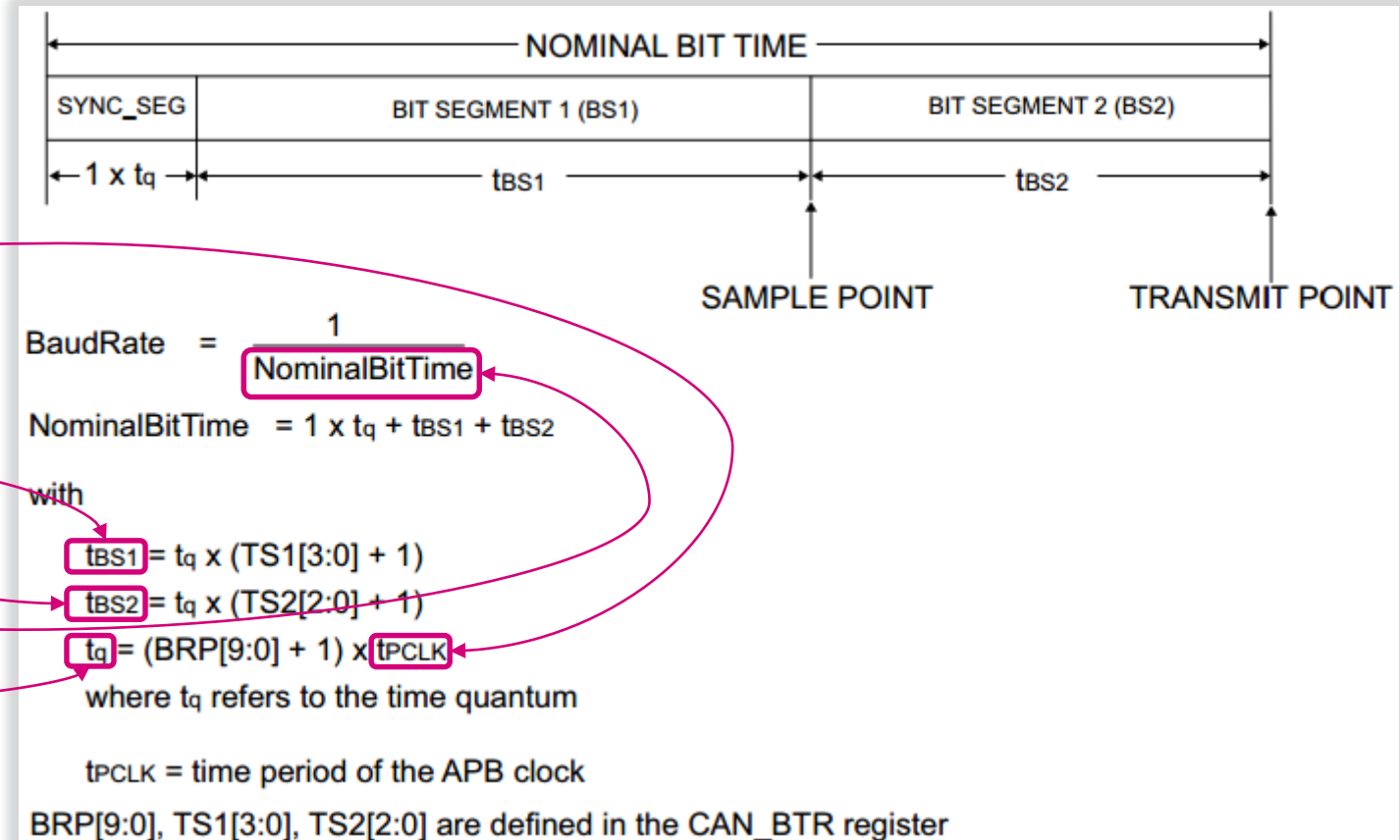
Operating Mode	Normal
----------------	--------

Restore Default Apply Ok Cancel

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

- Formula used to calculate CAN baudrate

Bit Timings Parameters	
Prescaler (for Time Quantum)	12
Time Quantum	250.0 ns
Time Quanta in Bit Segment 1	13 Times
Time Quanta in Bit Segment 2	2 Times
Time for one Bit	4000 ns



- More information:



RM0351

Reference manual

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

Easy configuration of CAN baudrate

- <http://www.bittiming.can-wiki.info/> webpage allows to obtain CAN baudrate configuration 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

ST Microelectronics bxCAN

Clock Rate: in MHz, from 1 to 300. Use the value of the clock rate at the first stage of the BaudRatePrescaler BTR, not the clock of the controller or crystal (typically for a 16 MHz clocked NXP SJA1000 use '8').

Sample-Point at: in %, from 50 to 90 (87.5 % is the preferred value used by CANopen and DeviceNet, 75% is the default value for ARINC 825).

SJW: numerical value from 1 to .. (1 is the preferred value used by CANopen and DeviceNet. The value is currently not used in all calculations, please look at the values used below the bit timing table).

Debug: ☐ generates debugging information to the calculation after the table.

Request Table

Example for
CAN baudrate
250 kbit/s

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

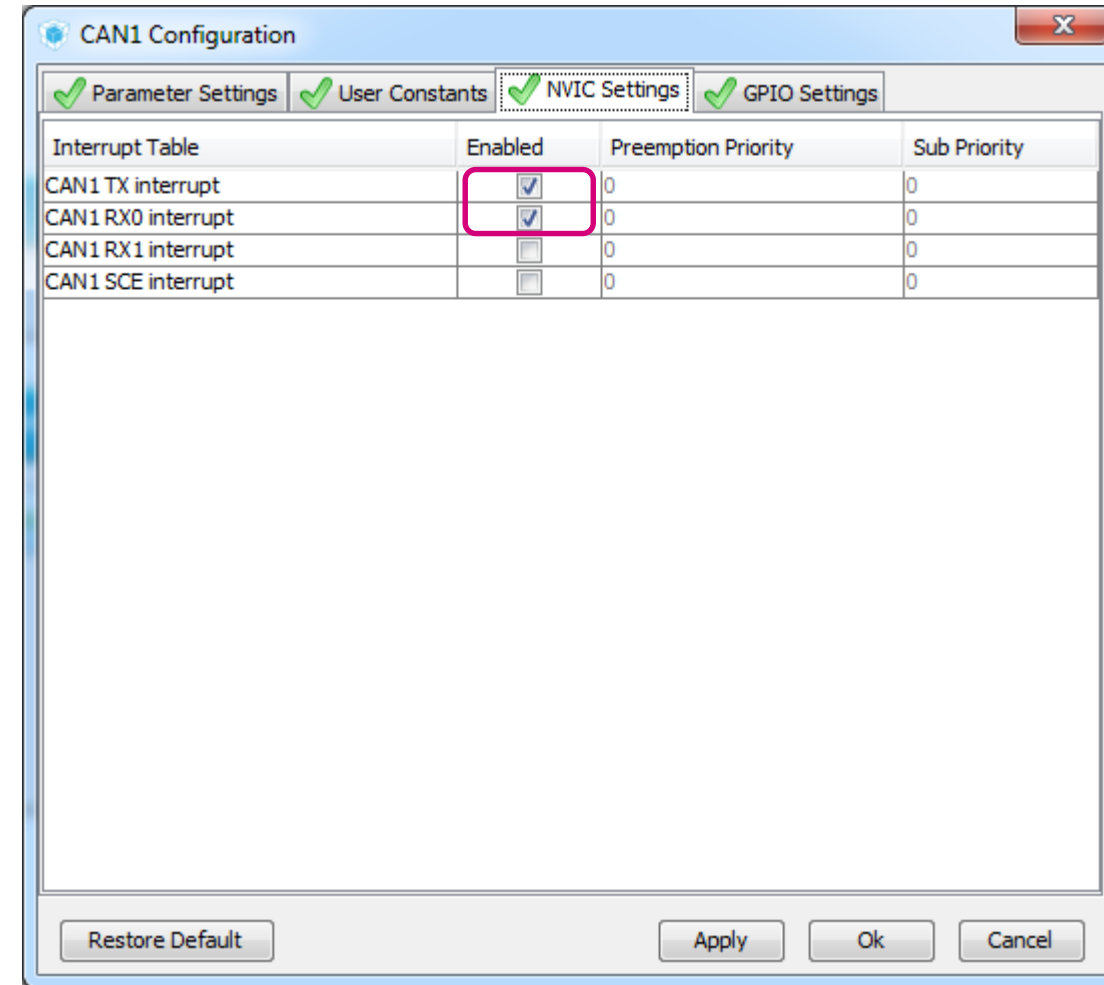
Bit Timings Parameters

Prescaler (for Time Quantum)	12
Time Quantum	250.0 ns
Time Quanta in Bit Segment 1	13 Times
Time Quanta in Bit Segment 2	2 Times
Time for one Bit	4000 ns

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

Project Settings

Project Code Generator Advanced Settings

Project Settings

Project Name
STM32L4_CAN

Project Location
C:\Users\szymon panecki\Desktop\ Browse

Toolchain Folder Location
C:\Users\szymon panecki\Desktop\STM32L4_CAN\

Toolchain / IDE
SW4STM32 ☒ Generate Under Root

Linker Settings

Minimum Heap Size 0x200

Minimum Stack Size 0x400

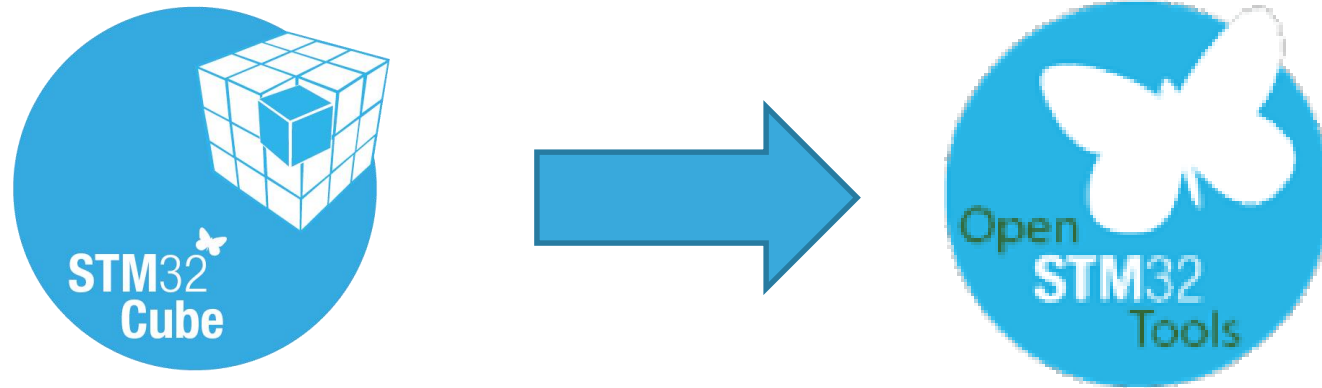
Mcu and Firmware Package

Mcu Reference
STM32L476VGTX

Firmware Package Name and Version
STM32Cube FW_L4 V1.6.0

☒ Use Default Firmware Location
C:/Program Files/STMicroelectronics/STM32Cube4.12/Libraries/STM32Cube_FW_L4_V1.6.0 Browse

Ok Cancel



- 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, transmission 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 structure items
configuration

CAN filter configuration
function call



Modifying the code

main.c file

Tasks:

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

CAN start

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

CAN notifications
(interrupts)

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

CAN TX message

```
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 */
```



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

The same application
using old CAN API

Modifying the code

data declaration - main.c file

Tasks:

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

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

/* USER CODE END PV */
```

CAN filter structure

CAN transmission and
reception structures

```
/* 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 structure 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;
```

CAN message structure items configuration

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

```
HAL_CAN_Receive_IT(&hcan1, CAN_FIFO0);
```

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

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

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

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

```
    HAL_CAN_Transmit(&hcan1, 10);
```

Call of function to send CAN message

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




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!



The image is a composite graphic. On the left, a man in a purple tank top and red pants is captured in a dynamic, acrobatic pose against a light blue background. In the center, a large green circle contains a white butterfly icon and the text "STM32 L4". To the right, a dark blue background features a network diagram with a central node showing a person jumping, connected to other nodes representing a cube, a microchip, and a smartphone. At the bottom, a grey bar contains three social media links: a Facebook icon followed by "/STM32", a Twitter icon followed by "@ST_World", and an "e2e" logo followed by "st.com/e2e".

 /STM32

 @ST_World

 st.com/e2e

www.st.com/mcu