CAN IMPLEMENTATION

“Working with CAN is not a **piece of CAKE**”

Interfacing of CAN protocol on **STM32** :

CAN implementation broadly divided into two:

1. Care to be taken on **Hardware** end
2. Care to be taken on **Software** end

1. Points to remember when dealing with hardware

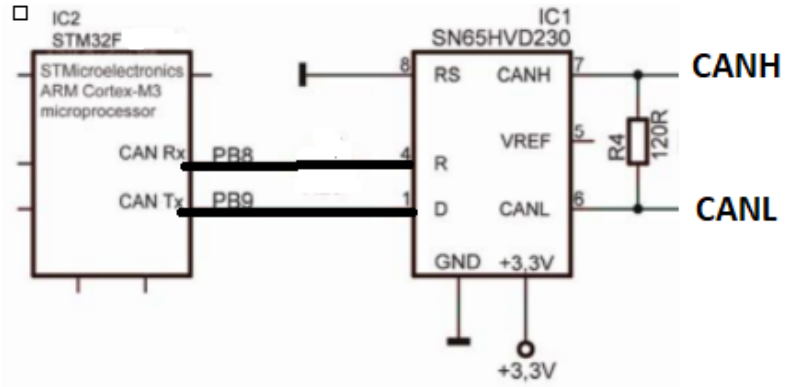
Hardware includes =>STM32 board, CAN transceivers, cables(End Point of Cables should be tight), Beagle Bone Black & Connections.

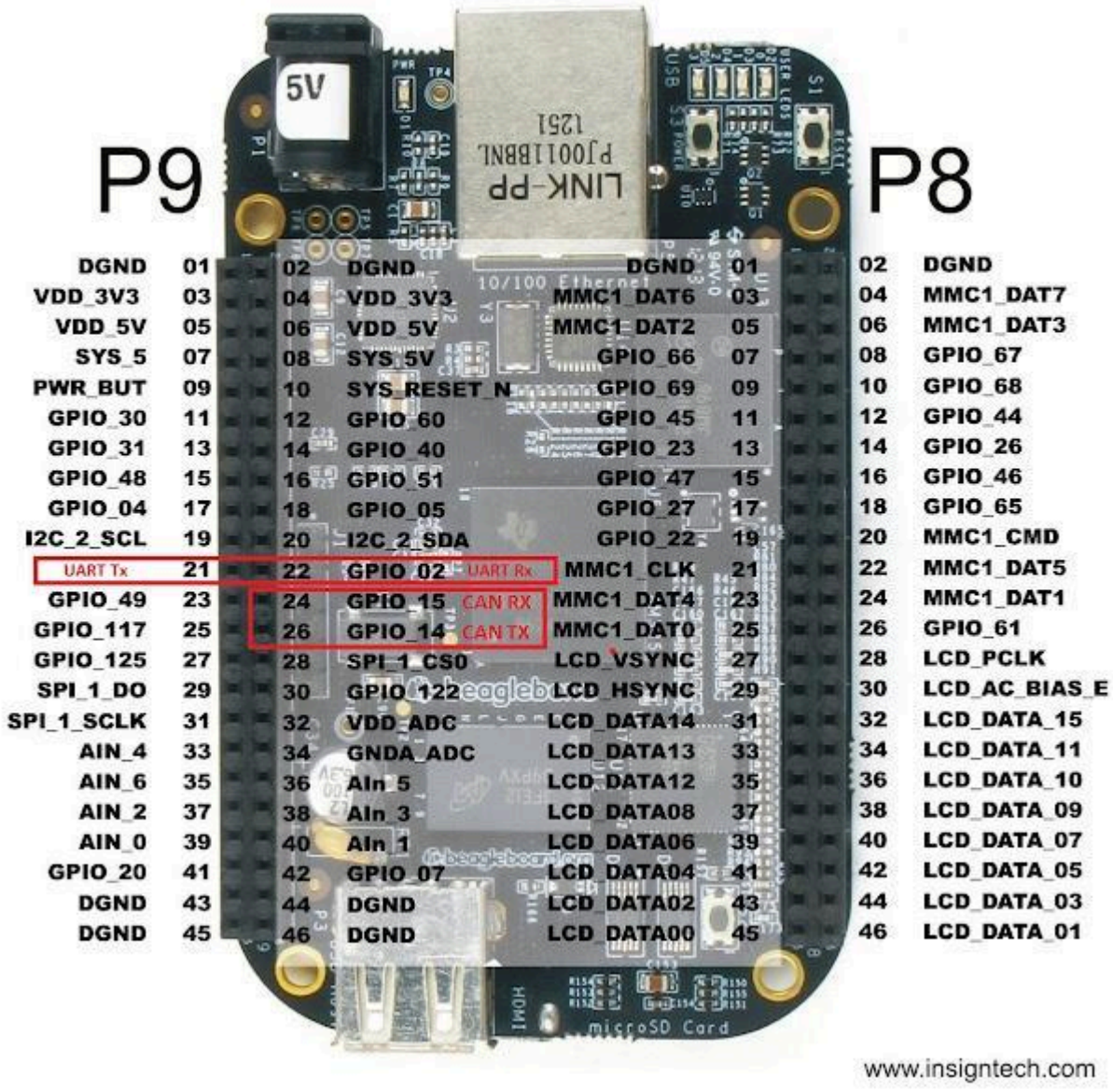
Connections:

STM32 → CAN TRX → CAN TRX → Beagle Bone

3.3V	3.3V			3.3V	VCC/3.3V
GND	GND			GND	GND
CAN_Rx	RX	CANH	CANH	RX	CAN_Rx
CAN_Tx	TX	CANL	CANL	TX	CAN_Tx

***Note: Also make sure to have a Common Ground for STM and BBB**





Note:-Refer the above pins for connection between Can transceiver to Beaglebone

- Working of CAN transceivers must be checked thoroughly.

  Operating voltage (**3.3V[SN65VD230]** or 5V), terminating resistor between CANH & CANL
  (120 ohms)

Purpose of 120 Ohm resistance is to avoid Reflections(due to Impedance Mismatching) in the Transmission Lines transmitting data at a high speed:

CANTx of STM -CANTx of Transceiver

CANRx of STM -CANRx of Transceiver

- Check if CANTx or CANRx of Transceiver is working fine.

In some cases CANTx of transceivers is not working but CANRx of transceivers is working.
Check the pins on IC or change the transceivers from another vendor and then check.

- **Connections must be tight while working with CAN** (minimum noise on CAN bus).

- CAN Transceivers must have proper soldering done (Check that No Joint should be dry soldered)

- Connections on breadboard must be tight (voltage properly applied).
- Directly connect the CANH & CANL using twisted pair cable & make sure the terminating resistors are present.
- Check the Silk Screen print on both the sides of IC is same .

2. Points to remember when dealing with Software part

Software includes=>STMCubeIDE

- SystemCore->RCC->HSE->Crystal/Ceramic Oscillator
- Clock Configuration(RCC external clock from oscillator) must be same at both Tx-Rx side

External clock on STM32F407VGTx HSE is on 8Mhz,

- Default Pins for CAN1 are PA11, PA12 as these Pins are Not Available as a Part of Headers Pins so Replace Them with PB8,PB9 (Beside Reset Button), to Replace PIN just do ***Ctrl + Click on the PA11,12 to show Alternate***
- CAN Configuration (**CAN1 is working on APB1 clock**, Prescaler, Time Quantas, Mode of operation) must be same, preferable clock on APB1 are **24Mhz(Preferred)**
- Preferable CAN protocol bit rate 62.5Kbps, 125Kbps[Speed upto 125kbps is Low speed CAN] & 500kbps[Speed upto 1Mbps is High Speed CAN].
- If you are using CAN Interrupt for receiving methods then don't forget to use Receive Callback, If you are using CANRx Polling don't forget to use RxFiFoFillLevel API.

Getting Started with CAN initialization:

CAN Bus has two initialization modes:

1. **Normal Mode:** for normal CAN Operation using at least two nodes
2. **Loopback Mode:** for Testing CAN BUS with single node

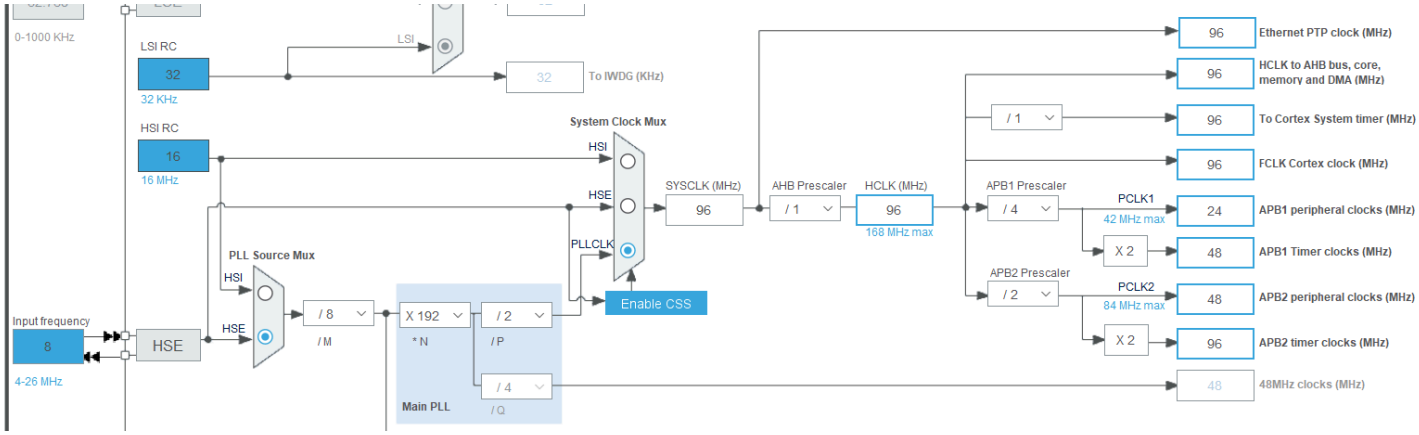
Note: In any of the above mode Transceiver must be attached.

Before interfacing CAN in Normal mode, check CAN in loopback mode i.e. transmit from CAN1 & receive on CAN1 internally in software.

1. Check the HAL_API version & level in STMCubeMx before generating code.

->This is the Clock Configuration to achieve **24 Mhz** at APB1

->External Oscillator is on 8 MHz, APB1 for CAN operates on 24Mhz.



2. CAN Initialization (Prescaler, Time Qantas, Mode Selection)-

Basically the CAN bit period can be subdivided into four time segments. Each time segment consists of a number of Time Quanta (tq).

The Time Quanta(Tq) is the smallest time unit for all configuration values.

#Configuring Pre Scalar for Tq

Example:

Preferred 1:

For Baud rate of 62.5Kbps if Pre-scalar = 24, Clock APB1=24Mhz -> 24Mhz/24 = 1Mhz=1us =1000ns

Preferred 2:

For Baud rate of 125Kbps if Pre-scalar = 12, Clock APB1=24Mhz -> 24Mhz/12 = 2Mhz=0.5us =500ns

Preferred 3:

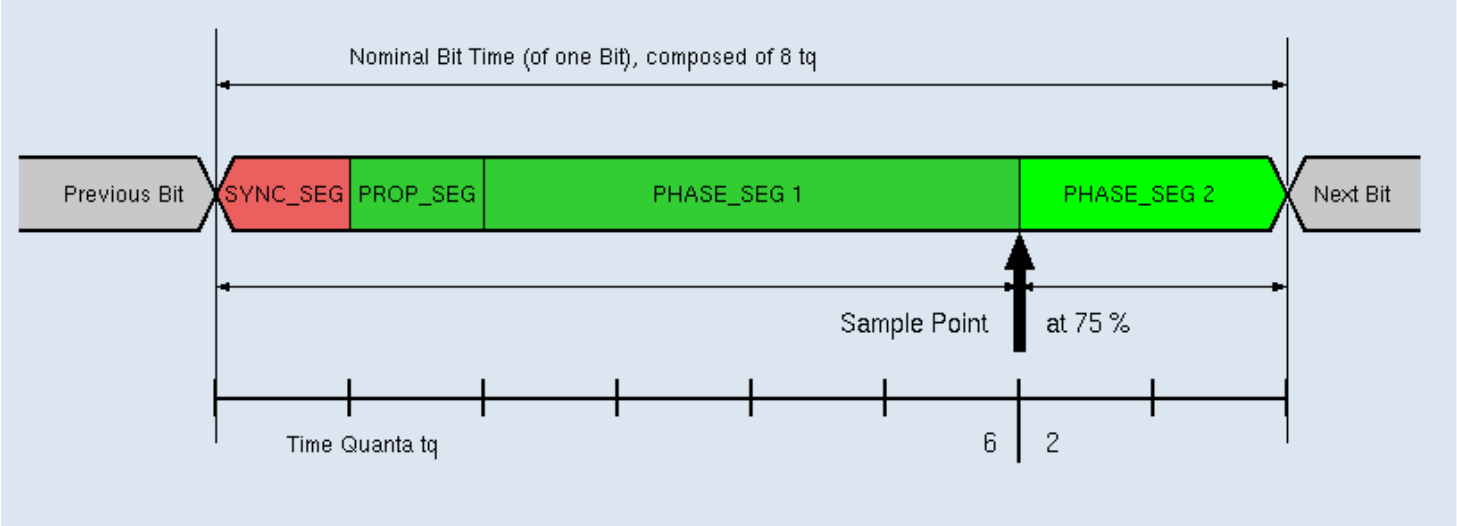
For Baud rate of 500Kbps if Pre-Scalar = 3, Clock APB1=24Mhz -> 24Mhz/3 = 8Mhz=0.125us=125ns

What will be the Tq for baud rate 250Kbps, 1Mbps?

- **SYNC_SEG** is 1 Time Quantum long. It is used to synchronize the various bus nodes.

- **PROP_SEG** is programmable to be 1, 2,... 8 Time Quanta long. It is used to compensate for signal delays across the network.
- **PHASE_SEG1** is programmable to be 1,2, ... 8 Time Quanta long. It is used to compensate for edge phase errors and may be lengthened during resynchronization.
- **PHASE_SEG2** (Seg 2) is the maximum of PHASE_SEG1 and the Information Processing Time long. It is also used to compensate for edge phase errors and may be shortened during resynchronization. For this the minimum value of PHASE_SEG2 is the value of SJW.

The picture shows you the time segments of a CAN-Bit as defined by ISO-11898.



Sync_Seg:	1 tq
Prop_Seg + Phase_Seg1:	1 .. 16 tq
Phase_Seg2:	1 .. 8 tq
(Table calculation uses Prop_Seg = 0)	

Bit Rate Calculation Examples

Q. Calculate Required Time Quanta to achieve different baud rate if the system clock is 24mhz and CAN Pre Scalar is 24 or 12 or 3.

Input Freq. at APB1 Bus	24mhz	24mhz	24mhz
Prescaler	24	12	3
CAN Clock(fcan)	24mhz/24=1mhz	24/12=2mhz	24/3=8mhz

Time Quanta(tq)=1/fcan	1/1mhz=1000ns	1/2mhz=500ns	1/8mhz=125ns
Bit rate	**62.5kbps**	**125kbps**	**500kbps**
Bit time =1/Bit rate	1/62.5kbps=16000ns	1/125kbps=8000ns	1/500kbps=2000ns
No of tq= Bit time/tq	16000ns/1000ns=16tq	8000ns/500ns=16tq	2000ns/125ns=16tq
TqSEG1	13	13	13
TqSEG2	2	2	2
ReSync	1	1	1

Q. Configure TSEG1 and TSEG2 to set sampling at 80% of a bit time.
(Tsync_seg+TSEG1)/(Tsync_seg+TSEG1+TSEG2)=80%

as we calculated above: bit time = Tsync_seg+TSEG1+TSEG2=16tq

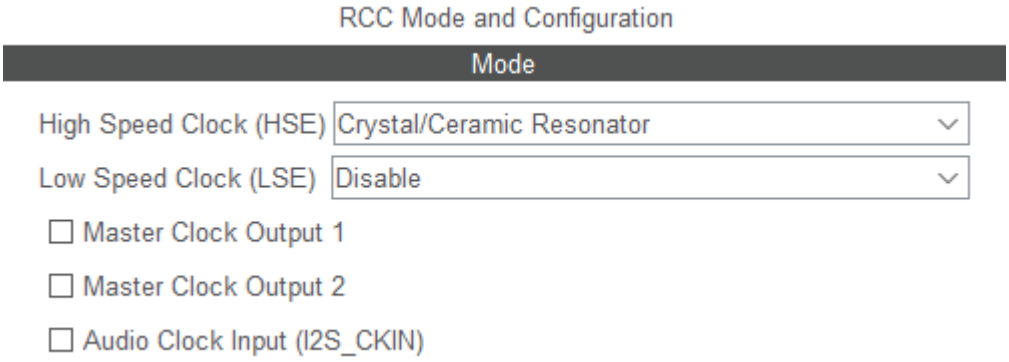
so , (1+TSEG1)/(16)=80%

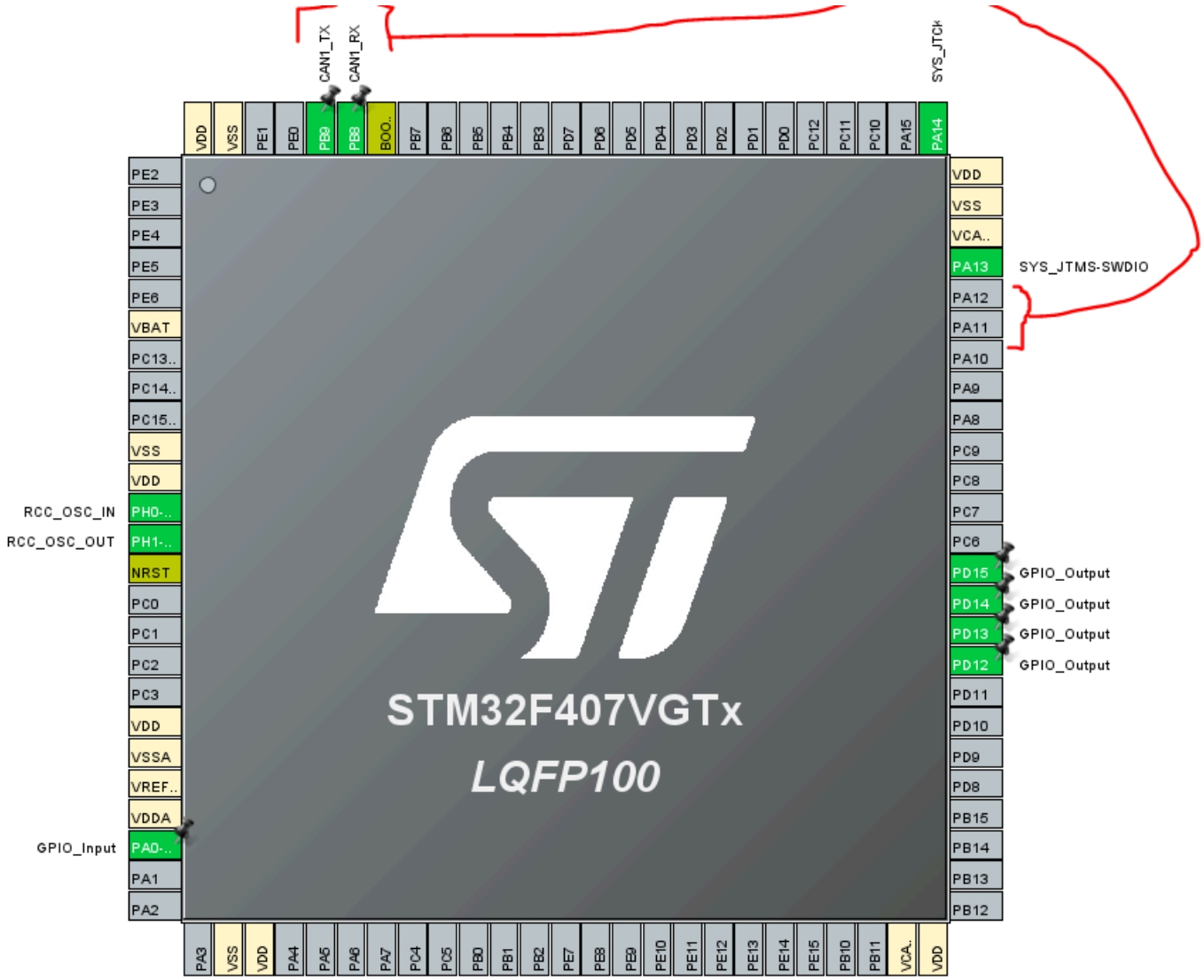
TSEG1=14-1=13

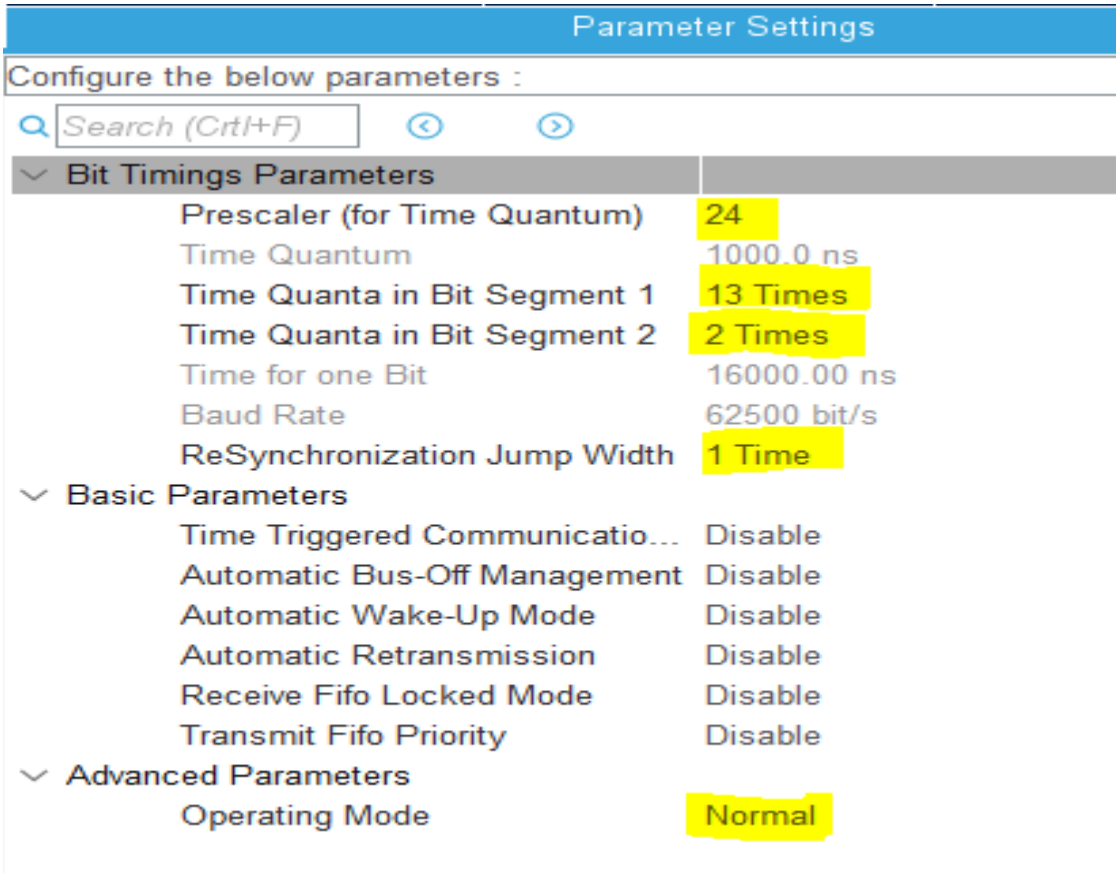
so TSEG2=20-1-15=2

ReSync=1 [Fixed]

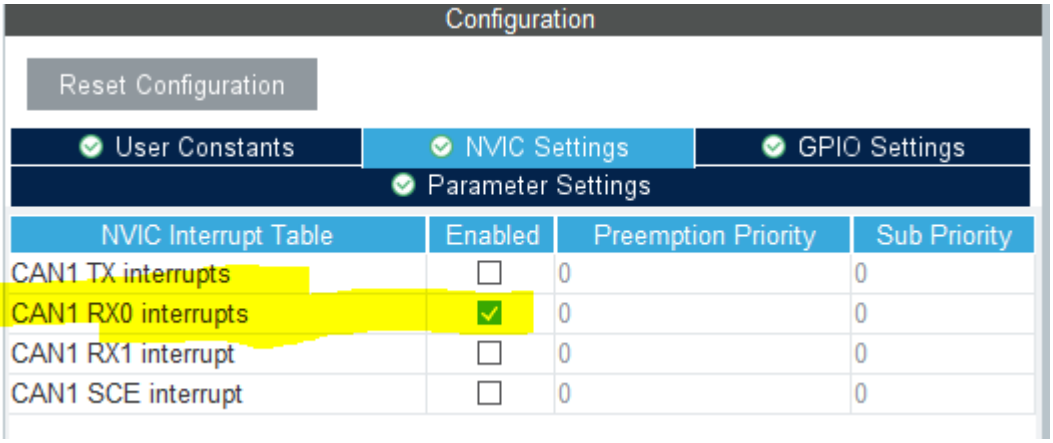
STM32CUBE IDE Settings

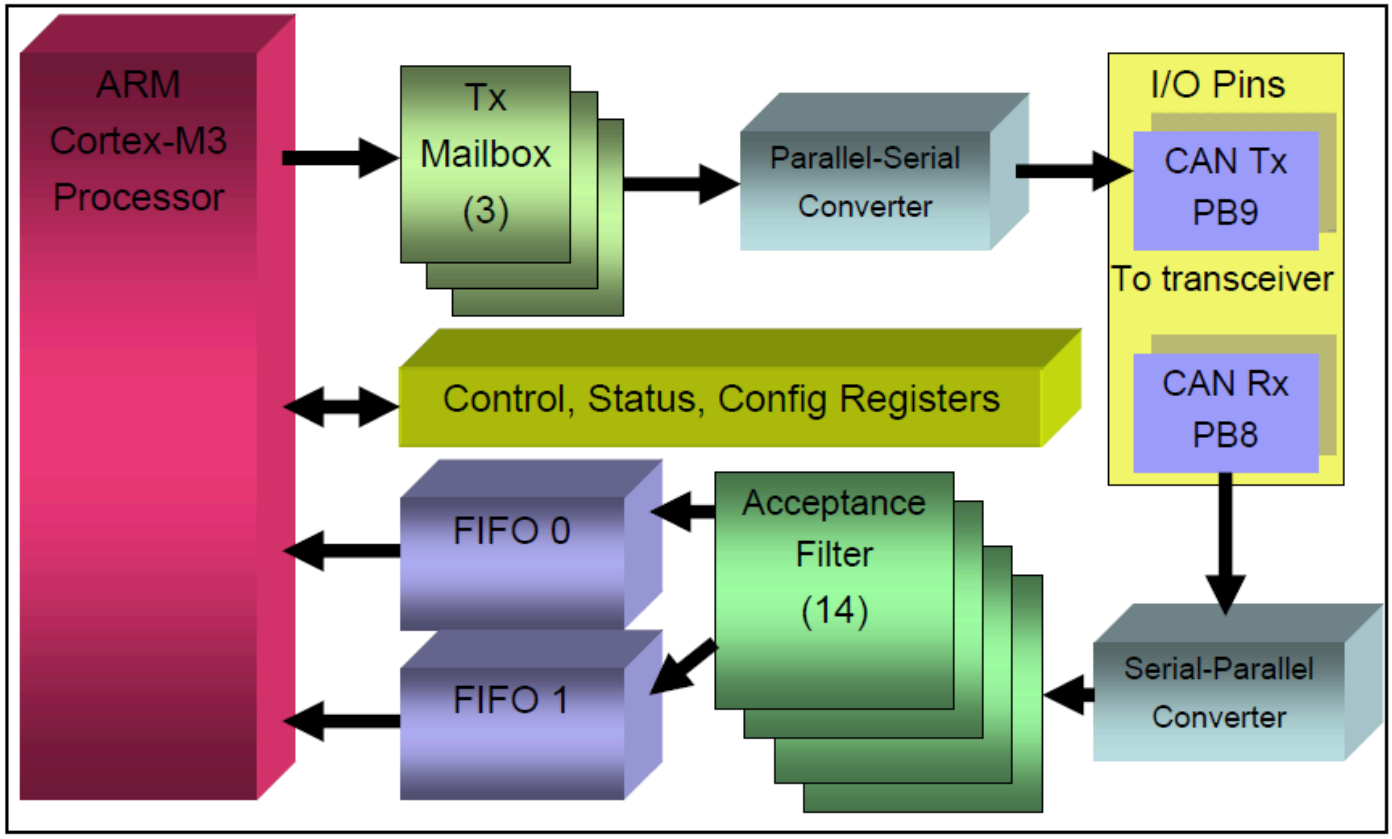



CAN1_TX
CAN1_RX
SYS_JTCK
SYS_JTMS-SWDIO
RCC_OSC_IN
RCC_OSC_OUT
GPIO_Input
GPIO_Output
GPIO_Output
GPIO_Output
GPIO_Output
STM32F407VGTx
LQFP100



3. Also Enable CANRx Interrupt & set subpriority-





4. Generate the code & start with CAN configuration-

- **Private Variables and Functions**

```
/* USER CODE BEGIN PV */
uint8_t ubKeyNumber = 0x0; //Counter Variable
CAN_TxHeaderTypeDef   TxHeader; // CAN Data Frame header fields like RTR Bit, DLC, ID
CAN_RxHeaderTypeDef   RxHeader; // CAN Data Frame header fields like RTR Bit, DLC, ID
uint8_t               TxData[8]; //Actual Payload
uint8_t               RxData[8]; //Actual Payload
uint32_t              TxMailbox; //Buffer for Tx Messages
/* USER CODE END PV */

/* USER CODE BEGIN PFP */
void CAN_filterConfig(void); //Acceptance filter Configuration
void LED_Display(uint8_t LedStatus);
/* USER CODE END PFP */
```

- **CAN Acceptance filter Configuration (filter bank, FilterIdHigh, FilterIdLow etc..)**
  - **This config will remain same for Polling and Interrupt**

```
//Can Filter config function to Accept All the message

static void CAN_filterConfig (void)                    //CAN Std ID = 11bits

{                                                      //Filter Scale is 32bits which is divided into
                                                       FilterIdHigh & FilterIdLow each 16bits
  CAN_FilterConfTypeDef  sFilterConfig;
```

```
  sFilterConfig.FilterNumber = 0;

  sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;

  sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;

  sFilterConfig.FilterIdHigh = 0x000 <<5;              //we want to fit CAN Std ID(11bits) in
                                                       FilterIdHigh(16bits) so <<5 is required
  sFilterConfig.FilterIdLow = 0x0000;                  //E.g. Std ID =0x123 ,if we shift it to <<5 it will
                                                       fit MSB of Std ID in 16bits of FilterIdHigh
  sFilterConfig.FilterMaskIdHigh = 0x0000; //0xFFE0    //0xFFE0 are High 11 bits of 16bit MaskIdHigh Reg.

  sFilterConfig.FilterMaskIdLow = 0x0000;              // if ((IdHigh & MaskIdHigh) == ID) then ACCEPT MSG;

  sFilterConfig.FilterFIFOAssignment = 0;              //Eg if((0x123<<5) & (0xFFE0) == 0x123) then 0x123
                                                       is ACEEPTED
  sFilterConfig.FilterActivation = ENABLE;             //Eg if((0x123<<5) & (0xFFE0) == 0x456) then 0x456
                                                       is REJECTED
  sFilterConfig.BankNumber = 14;


  if(HAL_CAN_ConfigFilter(&hcan1, &sFilterConfig) != HAL_OK)

  {

    /* Filter configuration Error */

    Error_Handler();

  }

}
```

CAN Polling

- CAN Start using HAL_APIs

```
* USER CODE BEGIN 2 */
  /*##Filter Configuration ########################################################*/
    CAN_filterConfig();

    /*##Start the CAN peripheral
###############################################*/
    if (HAL_CAN_Start(&hcan1) != HAL_OK)
        {
            /* Start Error */
```

```
            Error_Handler();
        }

    /*## Transmit Data ###########################################*/
    CAN_TransmitData();

    /*## Receive Data ############################################*/
    if(CAN_ReceiveData() == 1)
       {
         /* OK: Turn on LED1 */
               HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, SET);//green led
       }
       else
       {
         /* KO: Turn on LED2 */
               HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, SET);//orange led
       }
  /* USER CODE END 2 */
```

5. CAN Transfer & Receive:

- Use CAN Transmit & Receive APIs-

```
void CAN_TransmitData(void)
{
       /*##- Start the Transmission process ######################################*/
         TxHeader.StdId = 0x11;
         TxHeader.RTR = CAN_RTR_DATA;
         TxHeader.IDE = CAN_ID_STD;
         TxHeader.DLC = 2;
         TxHeader.TransmitGlobalTime = DISABLE;
         TxData[0] = 0xCA;
         TxData[1] = 0xFE;

         /* Request transmission */
         //API HAL_CAN_AddTxMessage to request a transmission of new Message
         if(HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox) != HAL_OK)
         {
           /* Transmission request Error */
           Error_Handler();
         }

         /* Wait for transmission to complete */
         //HAL_CAN_GetTxMailboxesFreeLevel To get no of free Tx Mailboxes, wait until at least 3
Mailbox is free
         while(HAL_CAN_GetTxMailboxesFreeLevel(&hcan1) != 3) {}
}

int CAN_ReceiveData(void)
{
       /*##- Start the Reception process ##########################################*/
       //Monitor the Reception of the Message using HAL_CAN_GetRxFifoFillLevel until at least one
Msg is Received
         if(HAL_CAN_GetRxFifoFillLevel(&hcan1, CAN_RX_FIFO0) != 1)
         {
           /* Reception Missing */
           Error_Handler();
         }

         //Get the Msg using HAL_CAN_GetRxMessage
         if(HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &RxHeader, RxData) != HAL_OK)
         {
```

```
          /* Reception Error */
          Error_Handler();
        }

        if((RxHeader.StdId != 0x11)                    ||
           (RxHeader.RTR != CAN_RTR_DATA)              ||
           (RxHeader.IDE != CAN_ID_STD)                ||
           (RxHeader.DLC != 2)                         ||
           ((RxData[0]<<8 | RxData[1]) != 0xCAFE))
               {
                    /* Rx message Error */
                    return 0;
               }

        return 1;

}
```

6. Error Handler:

```
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, SET);
  /* USER CODE END Error_Handler_Debug */
}
```

CAN Interrupt

Note: Sometimes **HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING)** won't allow the STM to be in the Debugging Session or Running the code, then you need to comment out this API and Run Code or Debug the code after that uncomment the API and Re-run the code or again Debug the Code then it works fine.[It's a Work Around]

```
/* USER CODE BEGIN 2 */

  /*##-Step1:Filter Configuration ########################################*/
      CAN_filterConfig();

  /*##-Step2:Start the CAN peripheral ####################################*/
      if (HAL_CAN_Start(&hcan1) != HAL_OK)
        {
             /* Start Error */
             Error_Handler();
        }
  /*##-Step3:Activate CAN RX notification i.e. INTERRUPTS ################*/
       if (HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING) !=
HAL_OK)
       {
         // Notification Error
         Error_Handler();
       }
```

```
      /*##-Step4:Configure Transmission process
#####################################*/
        TxHeader.StdId = 0x123;
        TxHeader.RTR = CAN_RTR_DATA;
        TxHeader.IDE = CAN_ID_STD;
        TxHeader.DLC = 2;
        TxHeader.TransmitGlobalTime = DISABLE;
  /* USER CODE END 2 */
```

Things to keep in while(1)

```
/* USER CODE BEGIN WHILE */
  while (1)
  {
          if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET)
                  {
             HAL_Delay(50);//Debouncing Delay
             if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET)
             {
                                 if (ubKeyNumber == 0x4)
                                        {
                                          ubKeyNumber = 0x00;
                                        }
                                 else
                                        {
                                          LED_Display(++ubKeyNumber);

                                          /* Set the data to be transmitted */
                                          TxData[0] = ubKeyNumber;
                                          TxData[1] = 0xAD;

                                          /* Start the Transmission process */
                                          if (HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData,
&TxMailbox) != HAL_OK)
                                                {
                                                     /* Transmission request Error */
                                                     Error_Handler();
                                                }
                           HAL_Delay(100);//Delay just for better Tuning
                                        }
                                }
                   }
    /* USER CODE END WHILE */
```

Callback for Interrupt

```
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
  /* Get RX message from Fifo0 as message is Pending in Fifo to be Read */
  if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, RxData) != HAL_OK)
  {
    /* Reception Error */
    Error_Handler();
  }

  /* Display LEDx */
  if ((RxHeader.StdId == 0x123) && (RxHeader.IDE == CAN_ID_STD) && (RxHeader.DLC == 2))
```

```
  {
    LED_Display(RxData[0]);
    ubKeyNumber = RxData[0];
  }
}

/**
  * @brief  Turns ON/OFF the dedicated LED.
  * @param  LedStatus: LED number from 0 to 3
  * @retval None
  */
void LED_Display(uint8_t LedStatus)
{
  /* Turn OFF all LEDs */
       HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  switch(LedStatus)
  {
    case (1):
      /* Turn ON LED1 */
             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
      break;

    case (2):
      /* Turn ON LED2 */
             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
      break;

    case (3):
      /* Turn ON LED3 */
             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
      break;

    case (4):
      /* Turn ON LED4 */
             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
      break;
    default:
      break;
  }
}
/* USER CODE END 4 */
```

CAN in RTOS

- Implement an **interrupt-driven** CAN BUS driver, as interrupt-based drivers are crucial when working with an RTOS.

- Create a task dedicated to handling CAN BUS messages, **including receiving,** decoding, and notifying other system components.

- This event-driven approach makes the system much more efficient.

- The interrupt signaling the reception of a new CAN frame can wake up a task to process the frame, eliminating the need for polling.

- Avoid polling for CAN messages, as it wastes CPU time and power when there's nothing to process.

- Organize the rest of your design into separate tasks, ideally by creating a task for each asynchronous input or related groups of inputs.

BeagleBone Black Enable Internet

Link

Configure the Windows firewall to allow pings

Link

Enable Internet Sharing Option Disabled by network Administrator

Link

BeagleBone Black Testing Virtual CAN Driver

```
modprobe vcan
ip link add dev vcan0 type vcan
ifconfig vcan0 up
ip -details -statistics link show vcan0
candump vcan0 &
cansend vcan0 442#DEADBABE
```

BeagleBone Black Enable Real CAN Driver

```
nano /boot/uEnv.txt
```

Edit :

```
###Overide capes with eeprom

uboot_overlay_addr0=/lib/firmware/BB-CAN1-00A0.dtbo
```

BeagleBone Black CAN Commands

```
modprobe can
modprobe can-raw
modprobe can-dev
ip link set can1 up type can bitrate 125000

candump can1 & // after running this connect stm32f4 disc board with transceiver & send data to BBB
//check weather data received and then proceed further , if not received data then debug connection/code here

cansend <interface_name> msg_id#data_in_hex_without_adding_0x
cansend can1 123#02AD  //this command will send 2 bytes of data ie 0x02 , 0xAD with id 0x123

ifconfig can1 down
ip -details -statistics link show can1
```

BeagleBone Black CAN Errors

A device may enter the “bus-off” state if too many errors occurred on the CAN bus. Then no more messages are received or sent. An automatic bus-off recovery can be enabled by setting the “restart-ms” to a non-zero value, e.g.:

```
sudo ip link set can1 type can restart-ms 100
```

Alternatively, the application may realize the “bus-off” condition by monitoring CAN error frames and do a restart when appropriate with the command:

```
sudo ip link set can1 type can restart
```

Note that a restart will also create a CAN error frame.

Python CAN in BBB

receiveCAN.py

```
#!/usr/bin/python

import can

def main():

        #can_interface = 'vcan0'

        can_interface = 'can1'

        bus = can.interface.Bus(can_interface, bustype='socketcan')

        try:

           while True:

             msg = bus.recv(0.0)

             if msg:

                msg_id=msg.arbitration_id

                print("ID",msg_id)

                data=msg.data

                data_format1=msg.data[1]

                print("Data Format0",data)

                print("Data Format1",data_format1)

                print("Data Format2",msg)


        except  KeyboardInterrupt:
```

```
                bus.shutdown()


if __name__ == "__main__":

        main()
```

sendCAN.py

```
import can

bustype = 'socketcan'

#channel = 'vcan0'

channel ='can1'


def main():

   bus = can.interface.Bus(channel=channel, bustype=bustype)

   for i in range(10):

      print ("Send a message...")

      #msg = can.Message(arbitration_id=0x123, data=[i, i+1, 0, 1, 3, 1, 4, 1], is_extended_id=False)

      #msg1 = can.Message(arbitration_id=0x123,data=[0x64, 0x65, 0x61, 0x64, 0x62, 0x65, 0x65, 0x66],
is_extended_id=False)

      #msg2 = can.Message(arbitration_id=0x123,data=b'deadbeef', is_extended_id=False)

      msg3 = can.Message(arbitration_id=0x123,data=b'de', is_extended_id=False)

      bus.send(msg3)

      time.sleep(1)




if __name__ == "__main__":

   main()
```

BeagleBone Black Enable UART2

```
nano /boot/uEnv.txt

Edit :

###Overide capes with eeprom

uboot_overlay_addr1=/lib/firmware/BB-UART2-00A0.dtbo
```

UART	RX	TX	Device
UART2	P9_22	P9_21	/dev/ttyO2

UART2 Enabled Confirmation

```
# cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pinmux-pins | grep -i uart

pin 84 (PIN84): 48024000.serial (GPIO UNCLAIMED) function pinmux_bb_uart2_pins group pinmux_bb_uart2_pins

pin 85 (PIN85): 48024000.serial (GPIO UNCLAIMED) function pinmux_bb_uart2_pins group pinmux_bb_uart2_pins
```

Install Minicom to use as UART Terminal

```
sudo apt-get update

sudo apt-get install minicom

sudo minicom -b 9600 -D /dev/ttyO2
```

Python UART Test Script

```
import Adafruit_BBIO.UART as UART

import serial


#UART.setup("UART2")


with serial.Serial(port = "/dev/ttyO2", baudrate=9600) as ser:

    print("Serial is open!")

    ser.write(b"Hello World123!")
```

More about UART

https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/uart

AWS CAN Script Publish and Subscribe

Link

Supported links:

https://www.youtube.com/watch?reload=9&v=ymD3F0h-ilE

https://www.youtube.com/watch?v=ar3I38lCLT4&list=PLfExI9i0v1sn_lQjCFJHrDSpvZ8F2CpkA&index=34

Time Quanta calculation- refer to this link

STMicroelectronics uses bxCAN, select the APB1 frequency & check the Time Quantas from the table.

http://www.bittiming.can-wiki.info/

CAN Primer -http://www.keil.com/download/files/canprimer_v2.pdf

Link for CAN Code:

This is correct that HAL CAN driver has been redesigned with new API.
Please find below a 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()

More complete 'how to use the new driver' is detailed in the driver header section itself.
https://community.st.com/s/question/0D50X00009XkgP6/halcaninit-is-failing