7. CAN bus communication

7.1. Purpose of the experiment

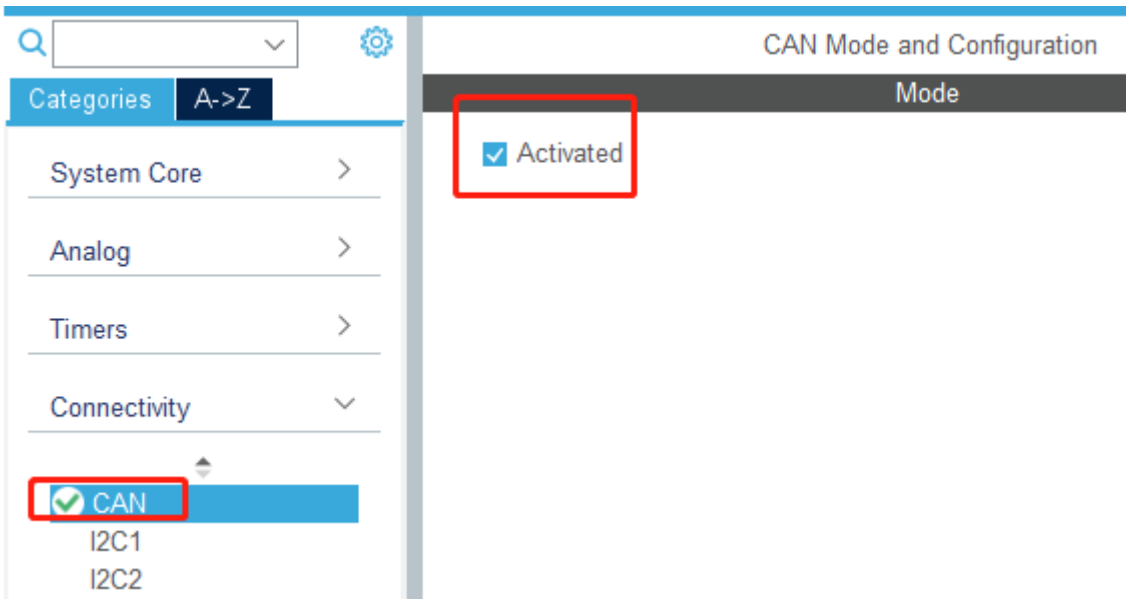
Using the CAN communication of STM32, using the loopback mode, the key control sends CAN data, interrupts the received CAN data and prints it out through the serial port assistant.

7.2. Configuration pin information

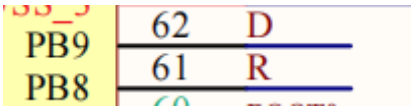
Since each new project needs configuration information, it is more troublesome. Fortunately, STM32CubeIDE provides the function of importing .ioc files, which can help us save time.

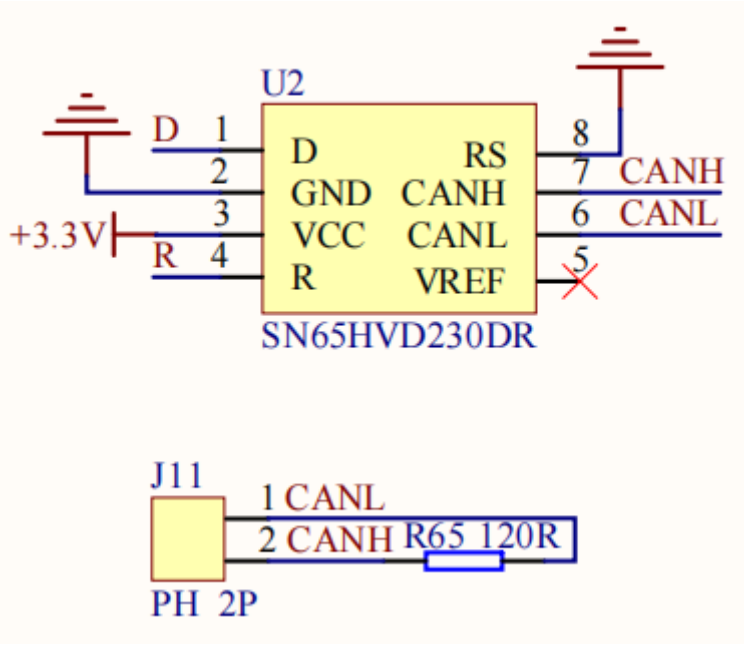
1. Import the ioc file from the Serial project and name it CAN.

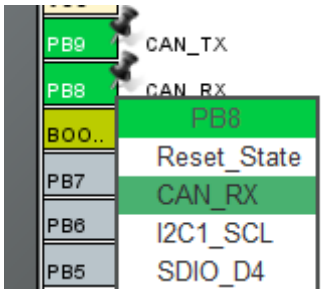
Find CAN in Connectivity and tick Activated to enable the CAN peripheral.



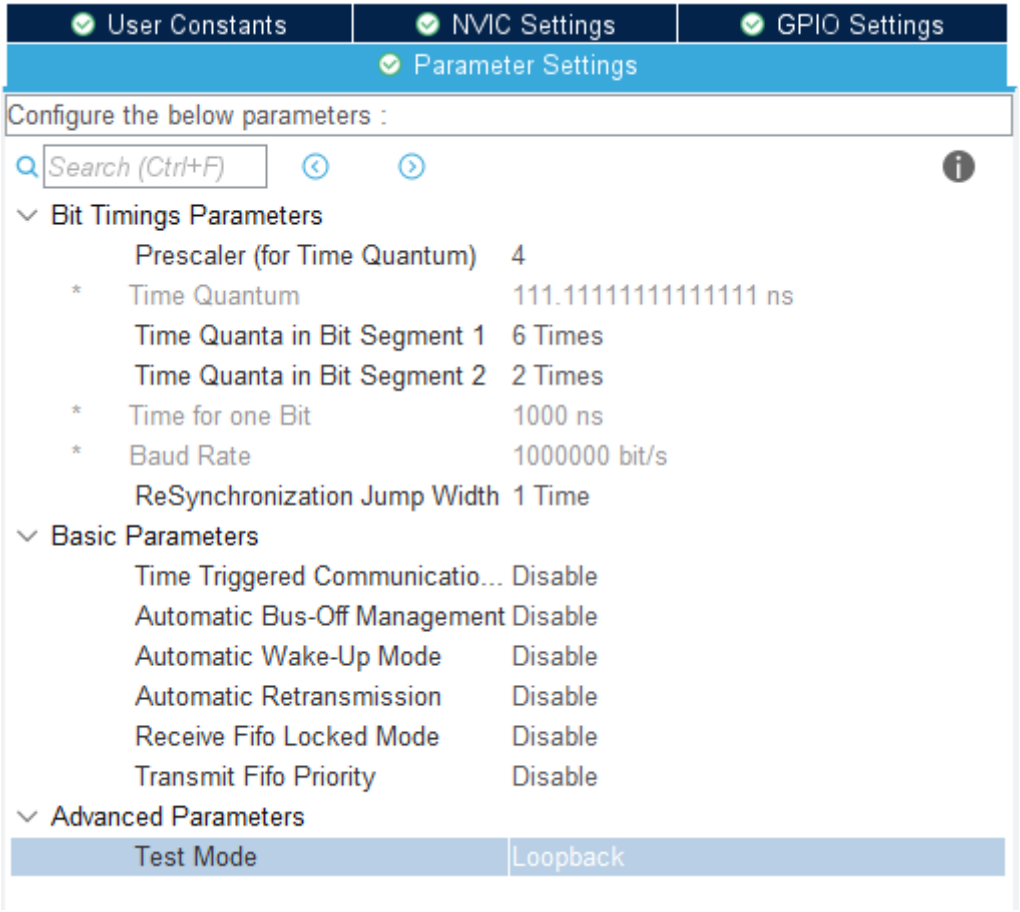
2. According to the schematic diagram, the pins connected to the CAN bus are PB8 and PB9, while the default CAN bus pins are PA11 and PA12, so it is necessary to manually modify the CAN bus pins to PB8 and PB9.



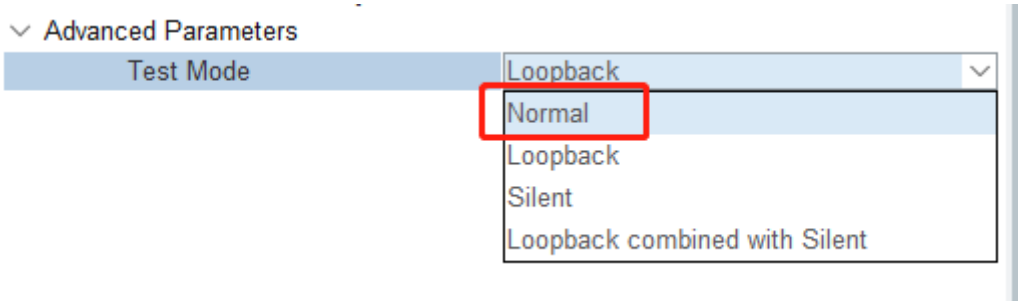




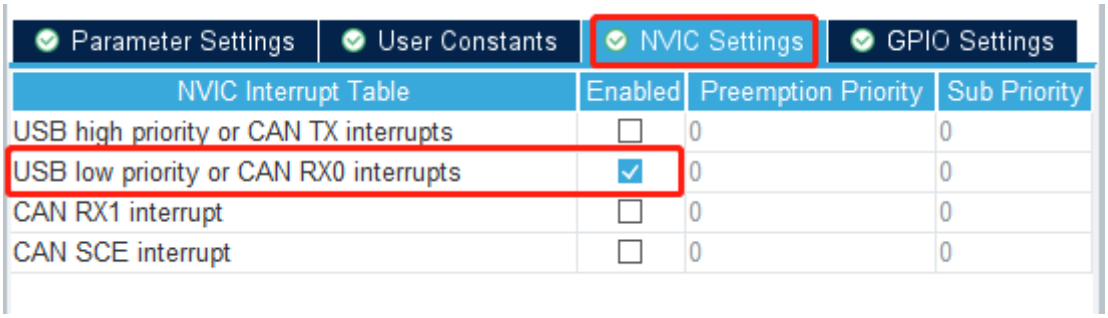
3. Set the parameters of the CAN peripheral, here we set the baud rate to 1000kbps and the mode to Loopback.



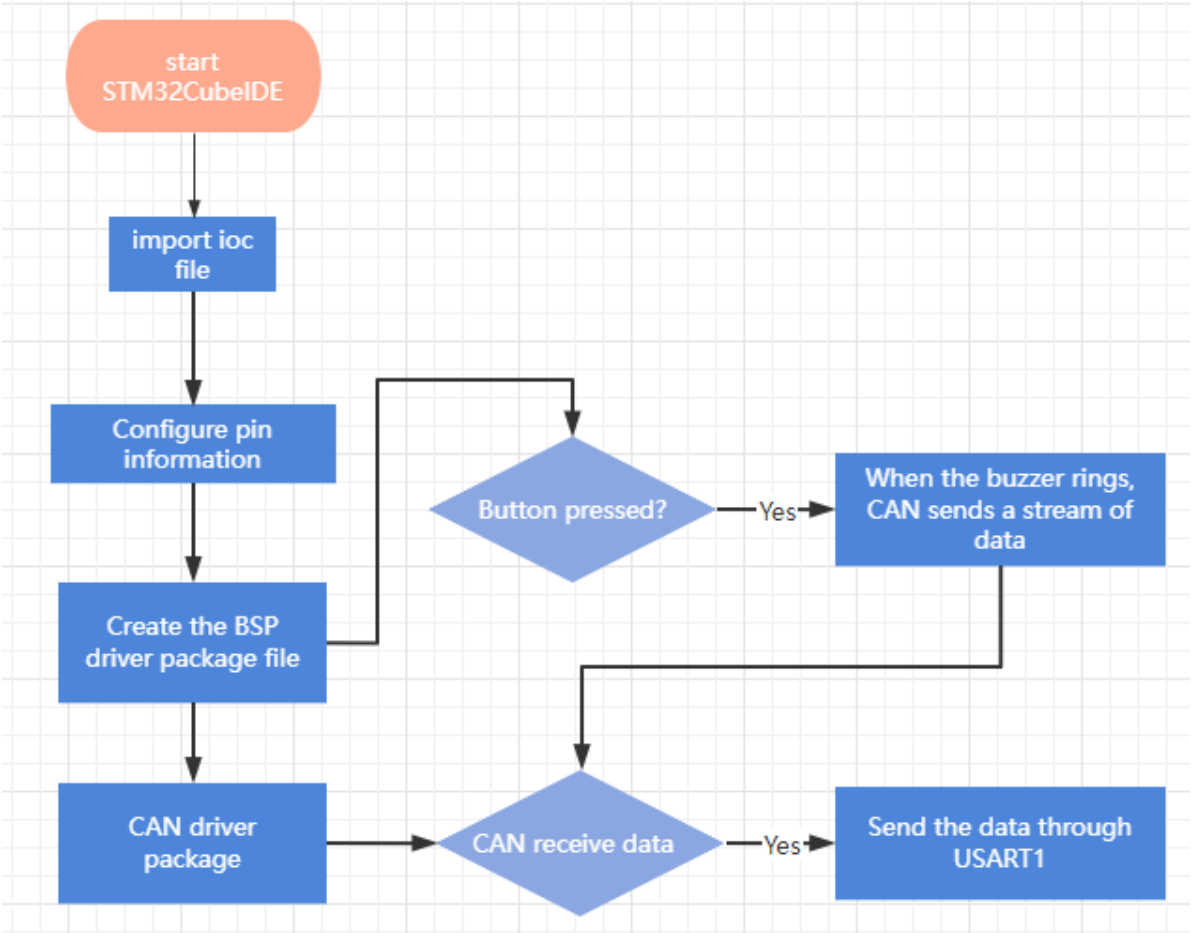
Since this is only used to test communication, choose Loopback mode (data is sent and received automatically); if you need to connect a third-party CAN device, please choose Normal mode (data receiving/sending independent).



4. Turn on the CAN RX0 interrupt in the interrupt setting. If the interrupt is not turned on, the data cannot be received.



7.3. Analysis of the experimental flow chart



7.4. core code explanation

1. Create new buzzer driver library bsp_can.h and bsp_can.c files in BSP. Add the following to bsp_can.h:

```
void Can_Init(void);
void Can_Test_Send(void);
```

2. Add the following in bsp_can.c:

Can_Init(): Initialize the CAN peripheral related content, set the CAN receive filter, and open the CAN bus communication.

```
// Initialize the CAN  初始化CAN
void Can_Init(void)
{
    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_FILTER_FIFO0;
    sFilterConfig.SlaveStartFilterBank = 27;
    sFilterConfig.FilterActivation = CAN_FILTER_ENABLE;

    // Setting the CAN Filter  设置CAN过滤器
    if (HAL_CAN_ConfigFilter(&hcan, &sFilterConfig) != HAL_OK)
    {
        Error_Handler();
    }

    // Start the CAN peripheral  启动CAN
    if (HAL_CAN_Start(&hcan) != HAL_OK)
    {
        Error_Handler();
    }

    // Activate CAN RX notification  启动CAN RX通知
    if (HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO0_MSG_PENDING) != HAL_OK)
    {
        Error_Handler();
    }
}
```

3. In order to test the sending data, the new Can_Test_Send() function sends the data through CAN and prints it to the serial port assistant. If you need to modify the sent data, you can modify the TxData array before sending.

```
// The test sends data through CAN  测试通过CAN发送数据
void Can_Test_Send(void)
{
    uint8_t TxData[8];
    uint32_t TxMailbox = 0;
    TxHeader.StdId = 0x000F;
    TxHeader.ExtId = 0x00;
    TxHeader.RTR = CAN_RTR_DATA;
    TxHeader.IDE = CAN_ID_STD;
    TxHeader.DLC = 8;
    TxHeader.TransmitGlobalTime = DISABLE;

    for (int i = 0; i < 8; i++)
    {
        TxData[i] = 1 << i;
    }
    printf("CAN Send:%02X %02X %02X %02X %02X %02X %02X %02X \n",
           TxData[0], TxData[1], TxData[2], TxData[3],
           TxData[4], TxData[5], TxData[6], TxData[7]);
    // Send Data  发送数据
    if (HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox) != HAL_OK)
    {
        Error_Handler();
    }
}
```

4. The CAN receive interrupt callback function prints the received CAN data through the serial port. The name of this function cannot be modified, otherwise the function cannot be called.

```
// CAN receives interrupt callbacks  CAN接收中断回调
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    if (hcan->Instance == CAN1)
    {
        uint8_t RxData[8];
        if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, RxData) != HAL_OK)
        {
            Error_Handler();
        }
        else
        {
            printf("CAN Receive:%02X %02X %02X %02X %02X %02X %02X %02X \n",
                   RxData[0], RxData[1], RxData[2], RxData[3],
                   RxData[4], RxData[5], RxData[6], RxData[7]);
        }
    }
}
```

5. In BSP initialization, call the Can_Init() function to initialize the CAN peripherals.

```
// The peripheral device is initialized  外设设备初始化
void Bsp_Init(void)
{
    Can_Init();
    USART1_Init();
    Beep_On_Time(50);
    printf("start\n");
}
```

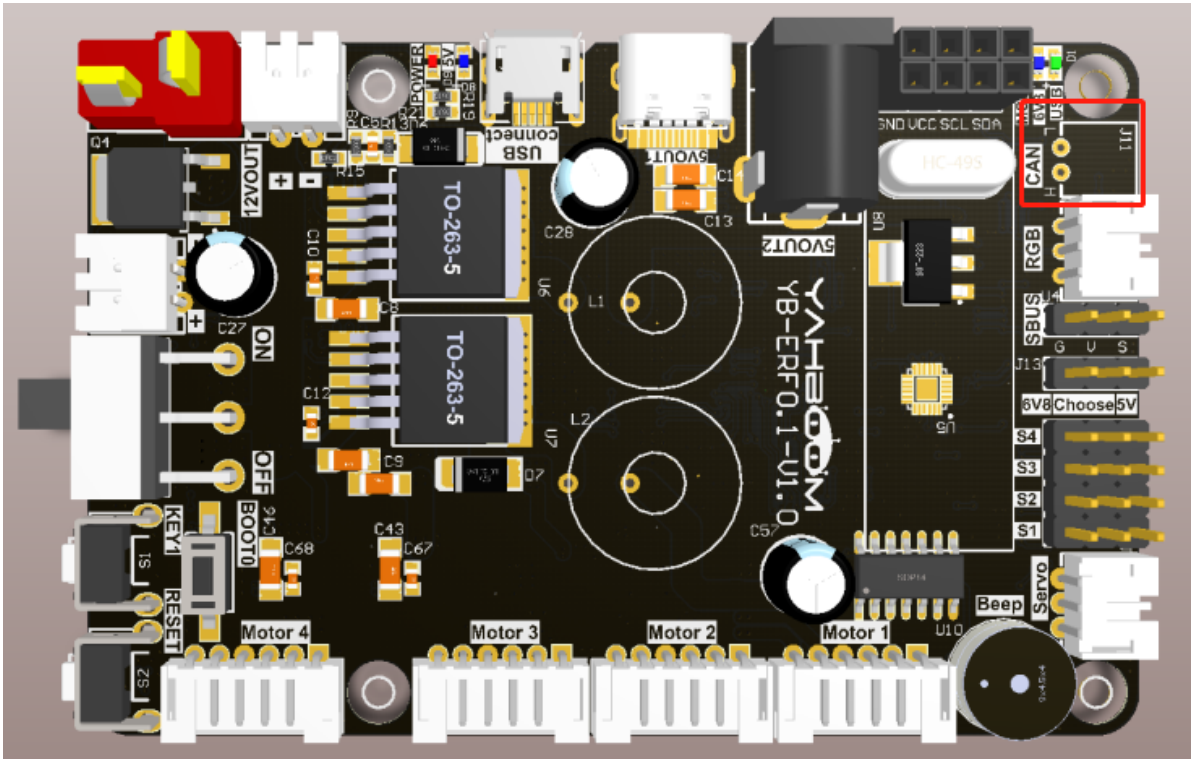
6. After the button is pressed, the function of sending CAN data is added.

```
// main.c中循环调用此函数，避免多次修改main.c文件。
// This function is called in a loop in main.c to avoid multiple modifications to the main.c file
void Bsp_Loop(void)
{
    // Detect button down events    检测按键按下事件
    if (Key1_State(KEY_MODE_ONE_TIME))
    {
        Beep_On_Time(50);
        Can_Test_Send();
    }

    Bsp_Led_Show_State_Handle();
    // The buzzer automatically shuts down when times out    蜂鸣器超时自动关闭
    Beep_Timeout_Close_Handle();
    HAL_Delay(10);
}
```

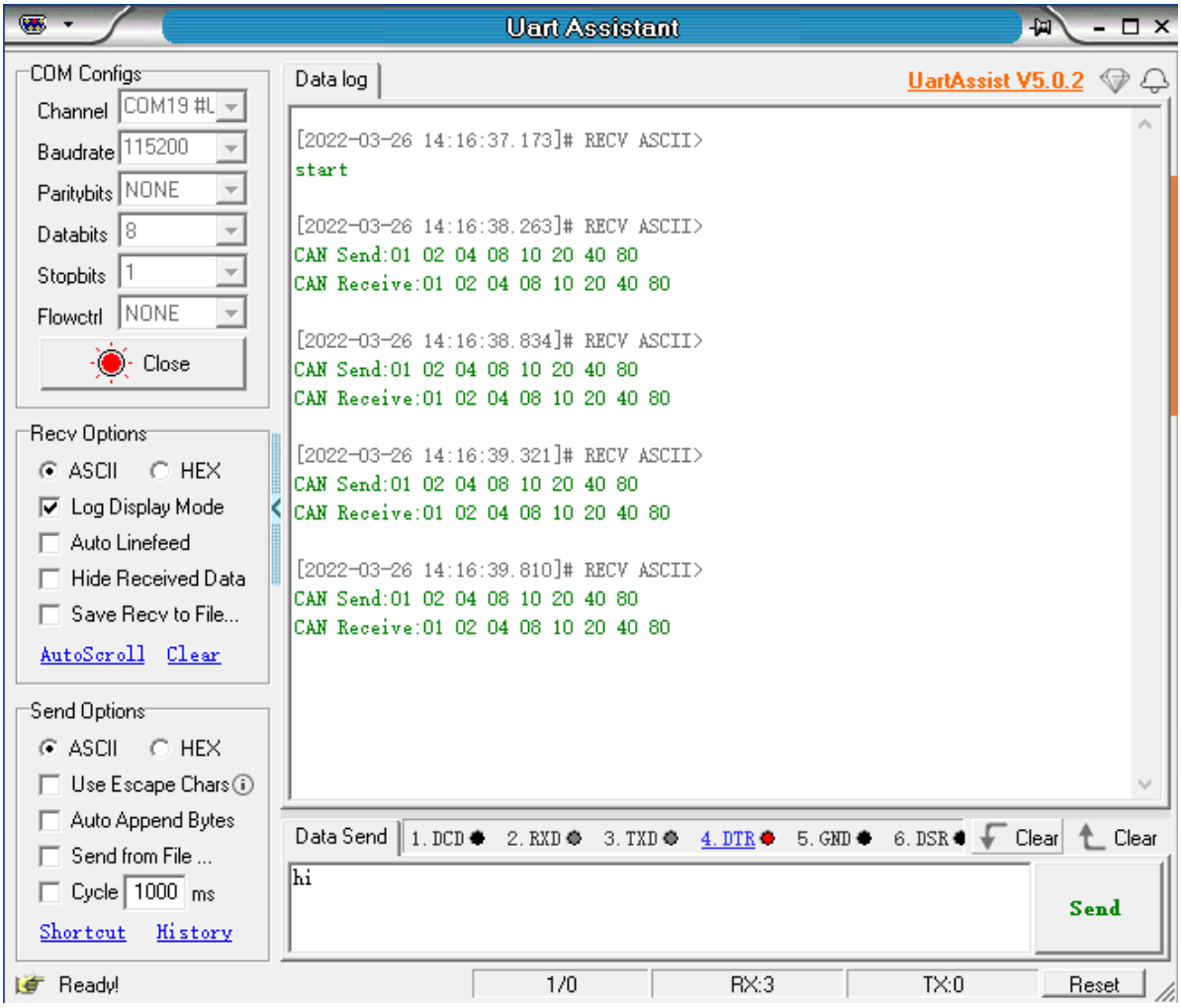
7.5. Hardware connection

Since the loopback mode is used, the CAN interface may not be connected to external devices.



7.6. Experimental effect

After programming the program, the LED light flashes once every 200 milliseconds. After connecting the expansion board to the computer through the micro-USB data cable, open the serial port assistant (the specific parameters are shown in the figure below), and the buzzer will sound every time the button is pressed. 50 milliseconds, you can see that the serial port assistant will display the data sent by CAN and the data received by CAN.


Uart Assistant
COM Configs
Channel COM19 #L
Baudrate 115200
Paritybits NONE
Databits 8
Stopbits 1
Flowctrl NONE
Close
Recv Options
ASCII HEX
Log Display Mode
Auto Linefeed
Hide Received Data
Save Recv to File...
AutoScroll Clear
Send Options
ASCII HEX
Use Escape Chars
Auto Append Bytes
Send from File ...
Cycle 1000 ms
Shortcut History
Data log
UartAssist V5.0.2
[2022-03-26 14:16:37.173]# RECV ASCII>
start
[2022-03-26 14:16:38.263]# RECV ASCII>
CAN Send:01 02 04 08 10 20 40 80
CAN Receive:01 02 04 08 10 20 40 80
[2022-03-26 14:16:38.834]# RECV ASCII>
CAN Send:01 02 04 08 10 20 40 80
CAN Receive:01 02 04 08 10 20 40 80
[2022-03-26 14:16:39.321]# RECV ASCII>
CAN Send:01 02 04 08 10 20 40 80
CAN Receive:01 02 04 08 10 20 40 80
[2022-03-26 14:16:39.810]# RECV ASCII>
CAN Send:01 02 04 08 10 20 40 80
CAN Receive:01 02 04 08 10 20 40 80
Data Send
1. DCD 2. RXD 3. TXD 4. DTR 5. GND 6. DSR
Clear
Clear
hi
Send
Ready!
1/0
RX:3
TX:0
Reset