7. CAN bus communication

7.1. Purpose of the experiment

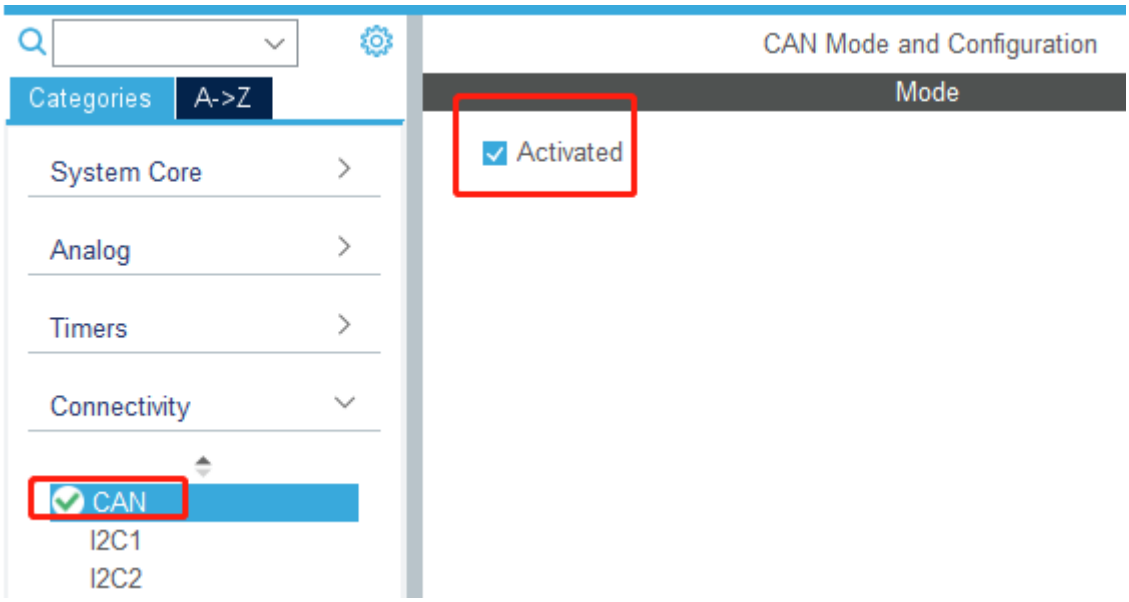
Use the CAN communication of STM32, using loopback mode, key control to send CAN data, interrupt to receive CAN data and print it out through serial assistant.

7.2 Configuring Pin Information

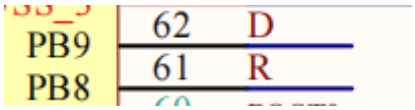
As we need to configure the information every time we create a new project, it is quite troublesome, good thing STM32CubeIDE provides the function of importing .ioc file, which can help us save time.

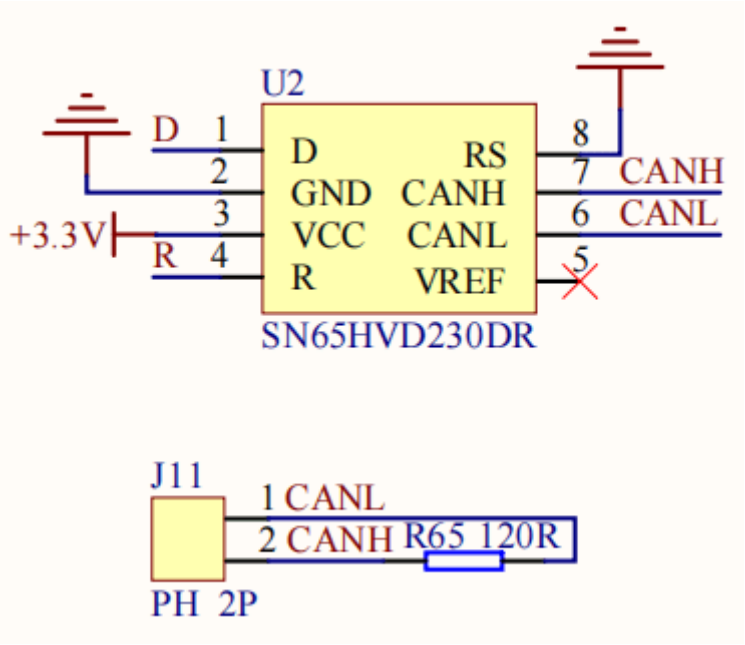
1. Import the ioc file from Serial's project and name it CAN.

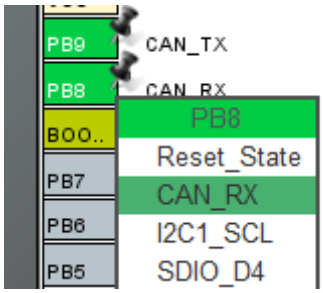
Find CAN in Connectivity and check Activated to enable CAN peripheral.



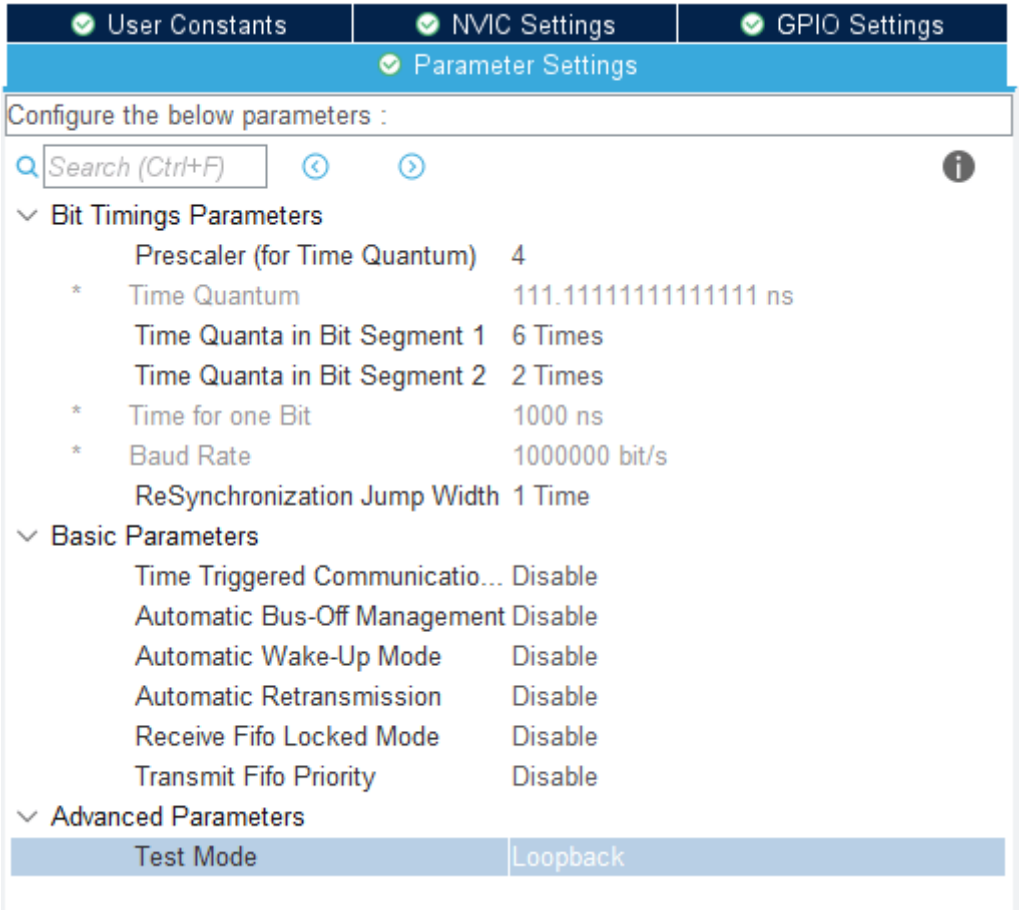
2. According to the schematic diagram, it can be seen that the pins connected to the CAN bus are PB8 and PB9, while the default CAN bus pins are PA11 and PA12, so you need to manually modify the CAN bus pins to PB8 and PB9.



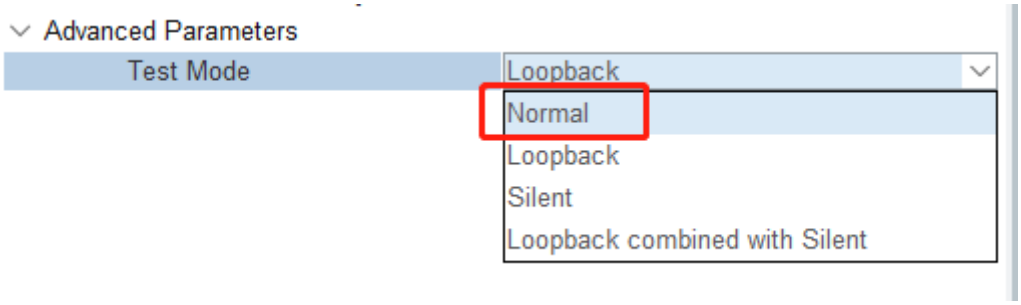




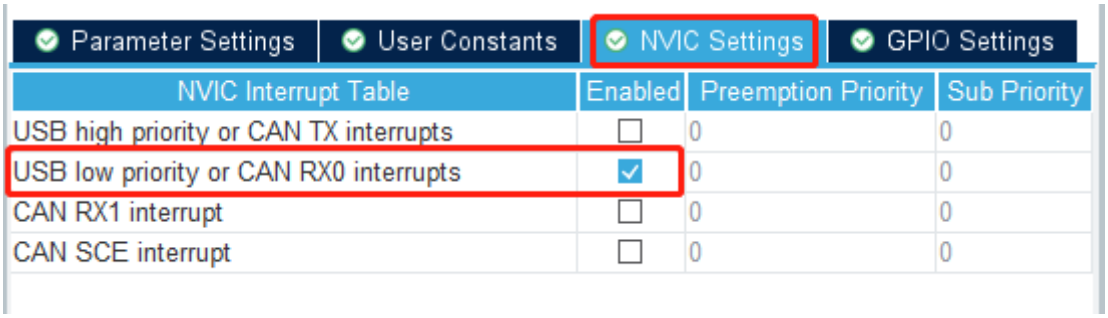
3. Set the parameters of the CAN peripheral, here we set the baud rate to 1000kbps and the mode to Loopback.



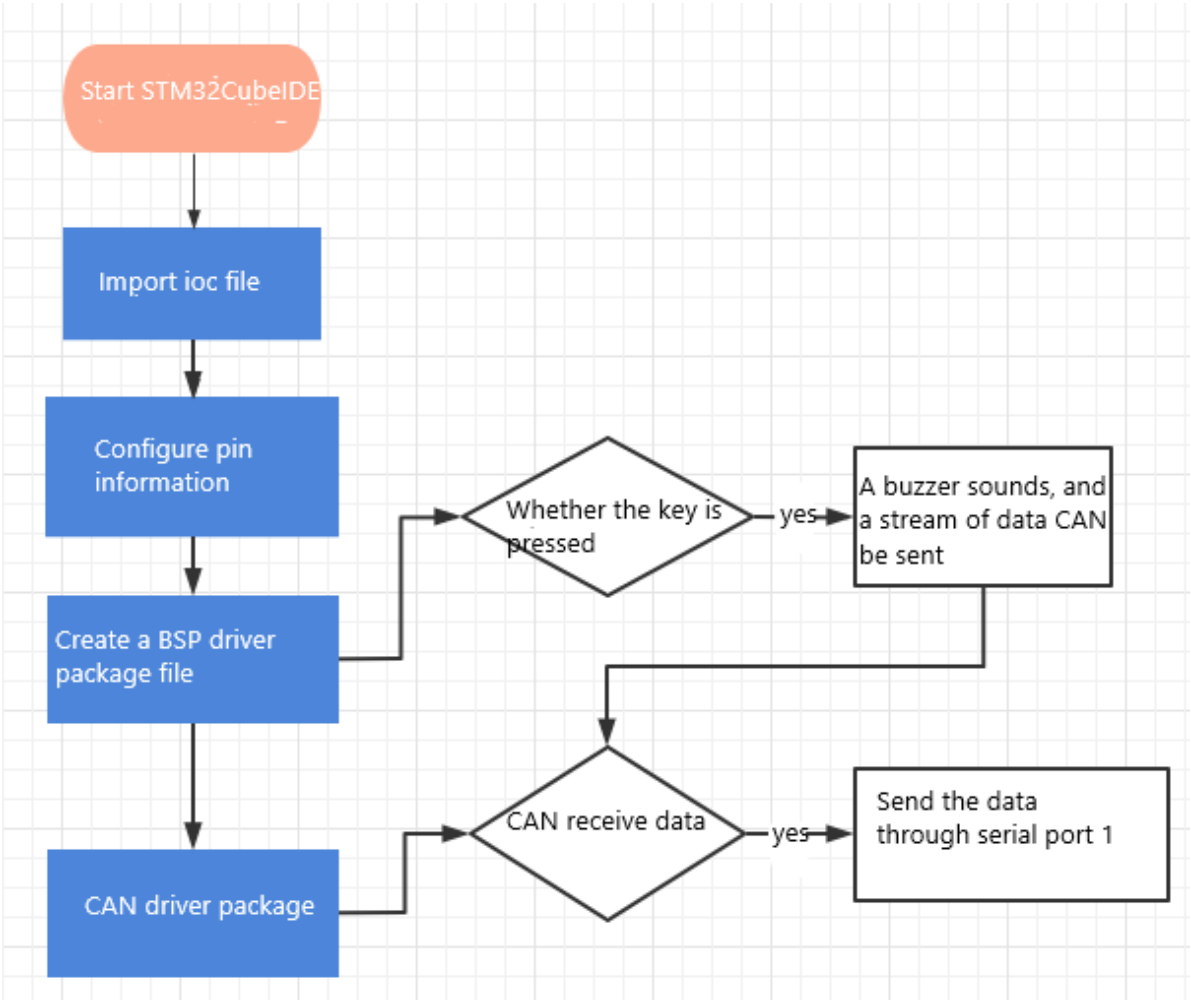
Since it is only used to test the communication here, Loopback mode (self-sending and self-collecting data) is selected; if you need to connect a third-party CAN device, please select the Normal mode (data receiving/transmitting independently).



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. Experimental flow chart analysis



7.4. Core Code Explanation

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

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

2. Add the following contents in bsp_can.c:

Can_Init():Initialize CAN peripheral related contents, set CAN receive filter, and open 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 sent data, create a new Can_Test_Send() function to send the data out via CAN and print it to the serial port assistant. If you need to modify the sent data, just 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. CAN receive interrupt callback function, will receive CAN data through the serial port print out. The name of this function can not be changed, otherwise this function can not 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 peripheral.

```
// 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");
}
```

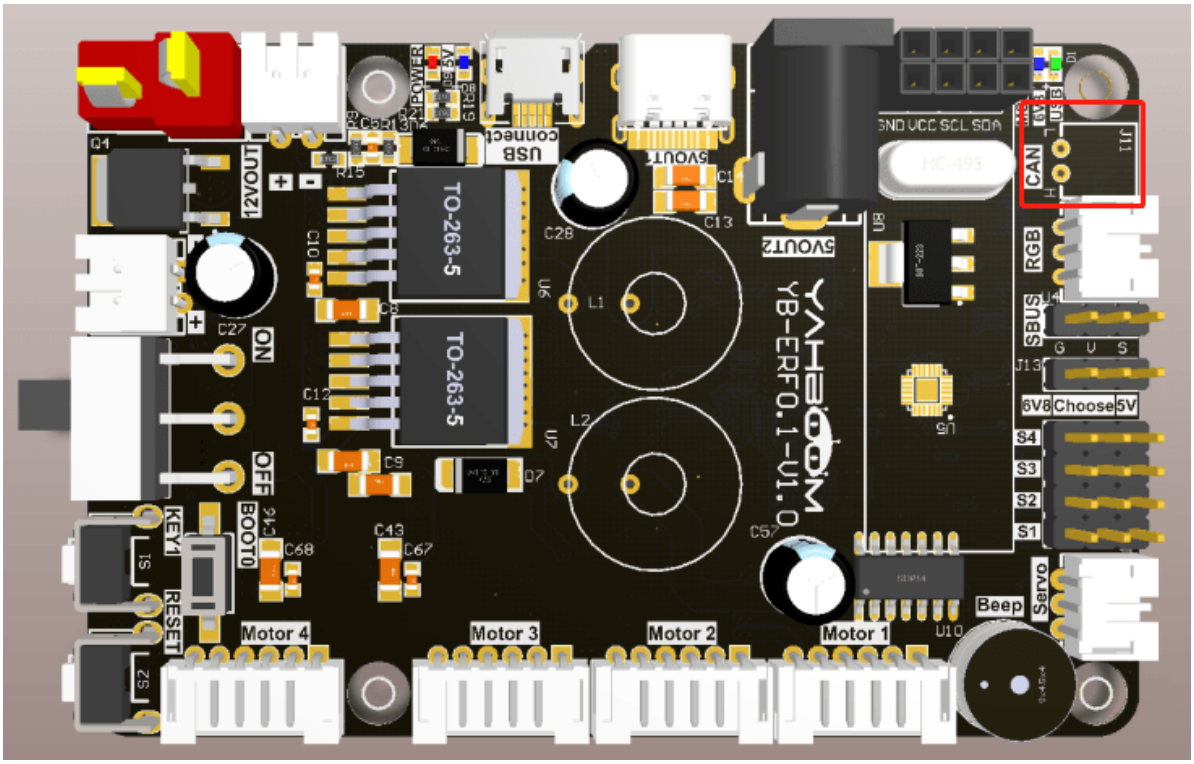
6. Add the function of sending CAN data after a key is pressed.

```
// 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);
}
```

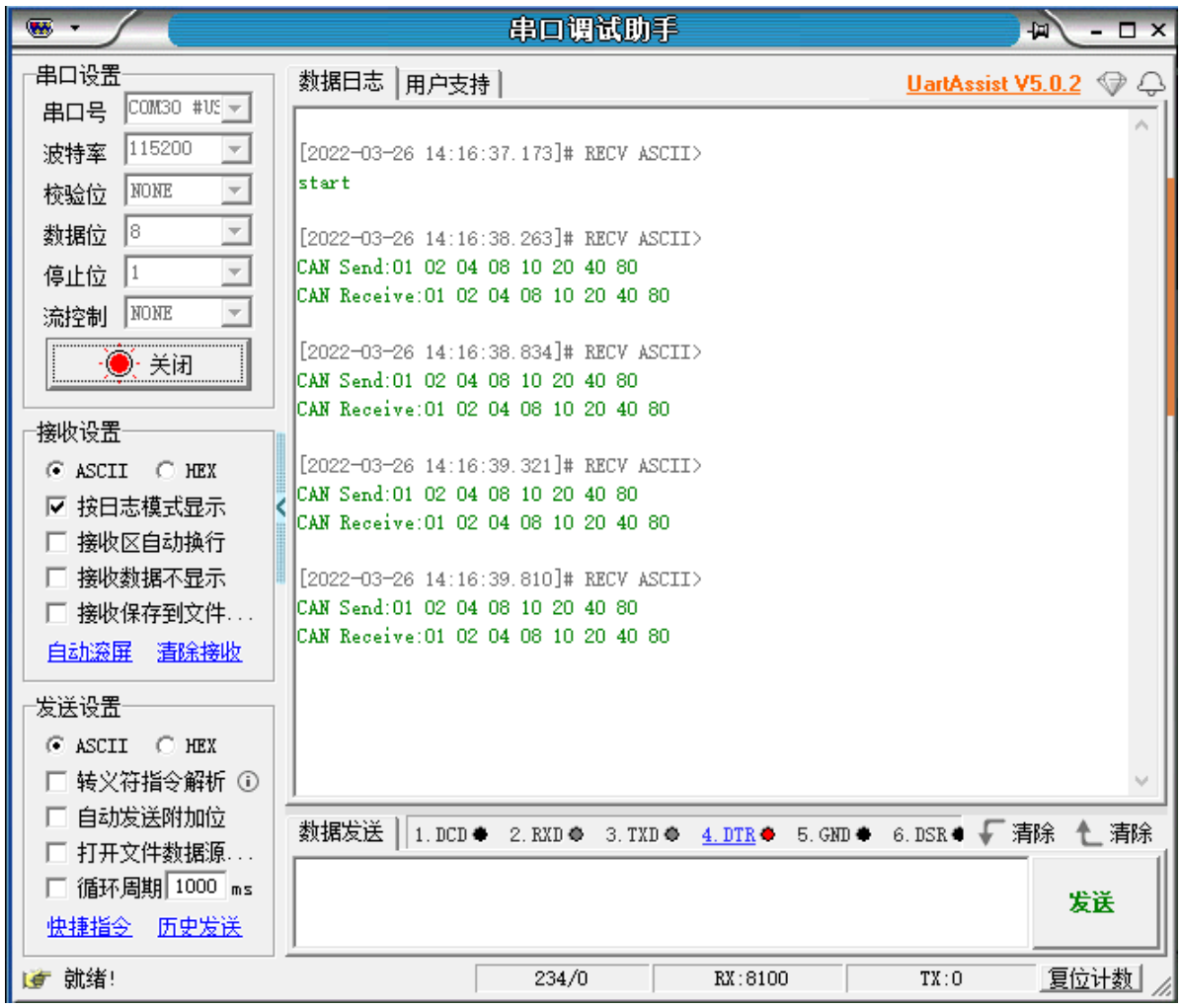
7.5. Hardware connection

Since the loopback mode is used, the CAN interface can be used without connecting an external device.



7.6 Experimental effect

After burning the program, the LED flashes every 200 milliseconds, connect the expansion board to the computer through the micro-USB cable and open the serial port assistant (the specific parameters are shown in the following figure), every time you press the key, the buzzer will sound for 50 milliseconds, and you can see that the serial port assistant displays the data sent by the CAN as well as the data received by the CAN.


串口调试助手
串口设置
串口号
COM30 #US
波特率
115200
校验位
NONE
数据位
8
停止位
1
流控制
NONE
关闭
接收设置
ASCII
HEX
按日志模式显示
接收区自动换行
接收数据不显示
接收保存到文件...
自动滚屏
清除接收
发送设置
ASCII
HEX
转义符指令解析
自动发送附加位
打开文件数据源...
循环周期
1000
ms
快捷指令
历史发送
数据日志
用户支持
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
数据发送
1.DCD
2.RXD
3.TXD
4.DTR
5.GND
6.DSR
清除
清除
发送
就绪!
234/0
RX:8100
TX:0
复位计数