BP15 系列 CAN 应用开发说明文档

V0.1

版本记录

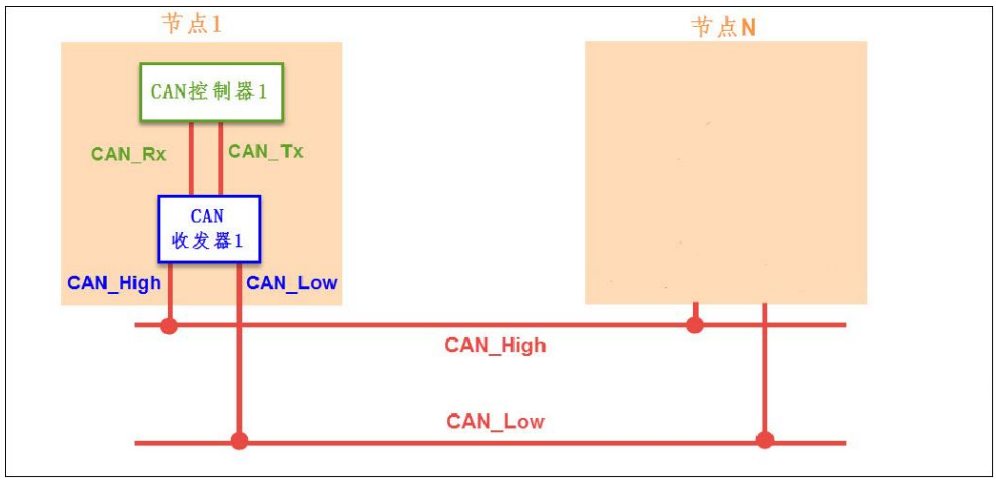
版本	作者	日期	修改日志
V0.1	Liangzx	2024-1-25	初版发布

目录

1. CAN 简介

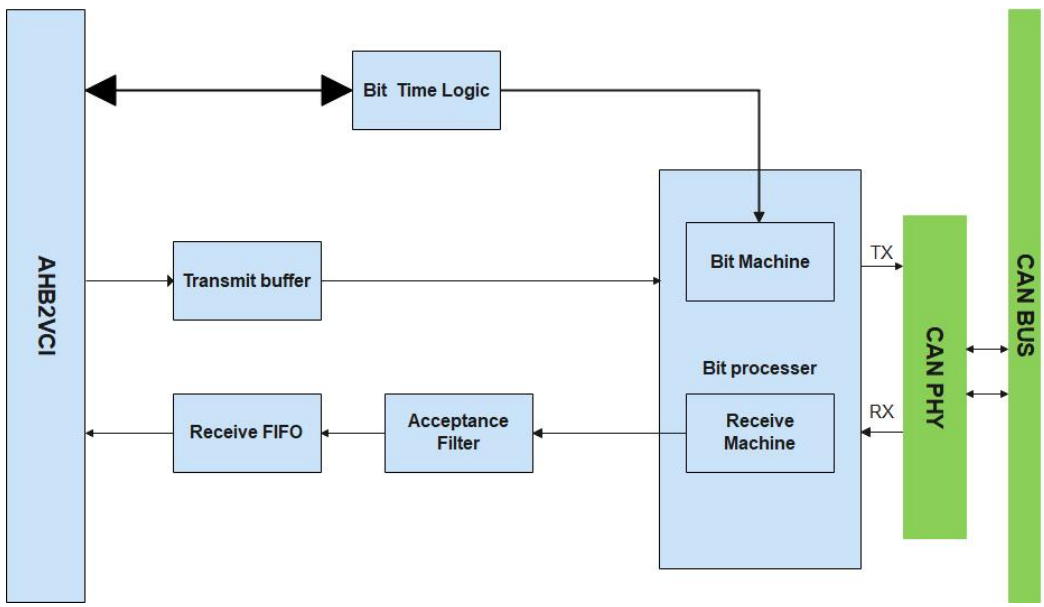
CAN(Controller Area Network)，是控制器局域网络的缩写，是 ISO 国际标准化组织的串行通信协议。CAN 具有很高的可靠性和良好的错误检测能力，广泛应用于汽车计算机控制系统和环境温度恶劣/电磁辐射强及振动大的工业环境。

CAN 总线上可以挂载多个通讯节点，节点之间的信号经过总线传输，实现节点间通讯。节点个数理论上不受限制，只要总线的负载足够即可，可以通过中继器增强负载。 CAN 通讯节点由一个 CAN 控制器及 CAN 收发器组成，控制器与收发器(电平转换)之间通过 CAN_Tx 及 CAN_Rx 信号线相连，收发器与 CAN 总线之间使用 CAN_High 及 CAN_Low 信号线相连。



CAN 是一种异步通信，只具有 CAN_High 和 CAN_Low 两条信号线，共同构成一组差分信号线，以差分信号的形式进行通讯。 CAN 控制器根据 CAN_High 和 CAN_Low 上的电位差来判断总线电平。总线电平分为显性电平（逻辑 0）和隐性电平（逻辑 1）。显性电平具有优先权，只要有一个单元输出显性电平，总线上即为显性电平。只有所有的单元都输出隐性电平，总线上才为隐性电平。

BP15 系列芯片内部集成了 1 个 CAN Controller，实现了 CAN 2.0 通信协议。支持 CAN2.0A 和 2.0B 协议规范，波特率最高 1Mbps。



2. CAN 报文（帧）CAN_DATA_MSG

2.1 CAN_DATA_MSG 结构体

```
typedef struct _CAN_DATA_MSG_
{
    uint32_t DATALENGTH      :  4;
    uint32_t RTR             :  1;
    uint32_t EFF             :  1;
    uint32_t Id;
    uint8_t  Data[8];
}CAN_DATA_MSG;
```

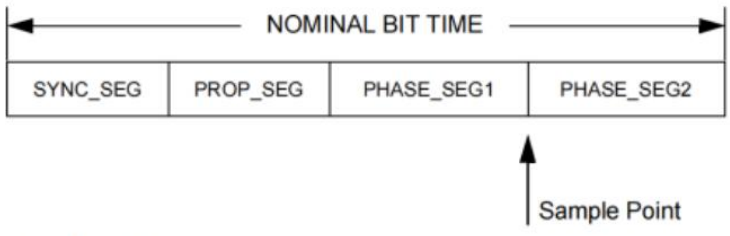
2.2 结构体成员说明

- DATALENGTH：报文数据长度，最长为 8 个字节。
- RTR：帧格式分为数据帧和远程帧。
  数据帧：RTR=0；
  远程帧：RTR=1，数据长度为 0。
- EFF：帧类型分为标准帧和扩展帧。
  标准帧：EFF=0，帧 ID 长度为 11Bit，范围为 0x00 - 0x7FF
  扩展帧：EFF=1，帧 ID 长度为 29Bit（11 位基本 ID + 18 位扩展 ID），范围为 0x00 - 0x1FFFFFFF
- Id：帧 ID，ID 越小在总线上传输的优先级越高。
- Data[8]：报文数据，长度由 DATALENGTH 控制。远程帧该数据是无效的。

3. CAN 波特率

3.1 CAN Bus Bit Timing

- NOMINAL BIT RATE
  理想情况下发送器在没有重新同步的情况下每秒发送的位数量。
- NOMINAL BIT TIME
  NOMINAL BIT TIME = 1 / NOMINAL BIT RATE
- Tq（时间单元）
  派生于振荡器周期的固定时间单元。
- BIT TIME
  CAN 协议把每一个 BIT TIME 分解成同步段(SS)，传播时间段(PTS)，相位缓冲段(PBS1 和 PBS2)。一个 BIT TIME 由 8~25 个 Tq 组成。



Partition of the Bit Time

SS(SYNC SEG)段：同步段，用于同步总线上不同的节点。若通讯节点检测到总线上信号的跳变被包含在 SS 段的 范围之内，则表示节点与总线的时序是同步的，当节点与总线同步时，采样点采集到的总线电平即可被确定为该位 的电平。SS 段固定大小为 1Tq。

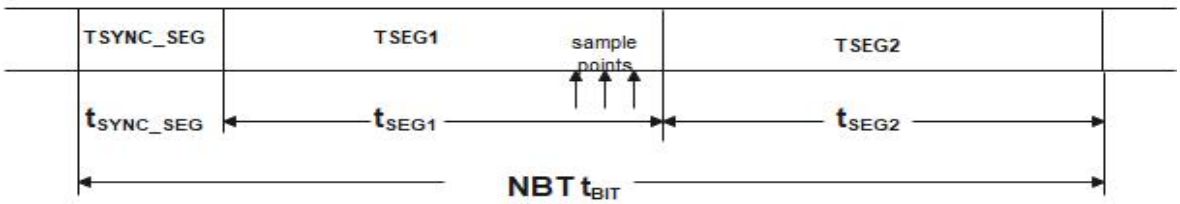
PTS(PROP SEG)段：传播时间段，用于补偿网络的物理延时时间。是总线上输入比较器延时和输出驱动器延时总 和的两倍。PTS 段的大小可以为 1~8Tq。

PBS1 段(PHASE SEG1)：相位缓冲段，用于补偿边沿阶段的误差，它的时间长度在重新同步的时候可以加长。PBS1 段的初始大小可以为 1~8Tq。

PBS2 段(PHASE SEG2)：相位缓冲段，用于补偿边沿阶段误差，它的时间长度在重新同步时可以缩短。PBS2 段的初始大小可以为 2~8Tq。

信号的采样点位于 PBS1 段与 PBS2 段之间，通过控制各段的长度，可以对采样点的位置进行偏移，以便准确地采样。

3.2 波特率计算



BRP --- NOMINAL BIT Rate Prescaler

NBT --- NOMINAL BIT TIME

BaudRate --- 波特率

F_{base} --- CAN 时钟频率

NBT = $T_{SEG1}+T_{SEG2}+T_{SYNC\_SEG}$

BRP = F_{base} **/ 2 * (**NBT * BaudRate) (NBT = 8~25 Tq)

$t_{clk} = 1/F_{base}$

$Tq = 2 * t_{clk} * (BRP + 1)$

$t_{syncseg} = 1 * Tq$

$t_{tseg1} = Tq * (TSEG1 + 1)$

$t_{tseg2} = Tq * (TSEG2 + 1)$

SJW -- Synchronization Jump Width

采样点可以由 SJW 进行设置。SJW 定义了 Tq 时钟周期的最大比特数，可以缩短或延长，以实现对总线上的数据转换的重新同步。

0：1 Tq clock cycle

1：2 Tq clock cycles

2：3 Tq clock cycles

3：4 Tq clock cycles

比如：设置波特率 100Kbps，F_{base} =24M

T_{SEG1} = 13

T_{SEG2} = 6

T_{SYNC_SEG} = 1

NBT = $T_{SEG1}+T_{SEG2}+T_{SYNC\_SEG}$ = 13+6+1=20

BRP = 24M/(2*100K*20) = 6

所以寄存器设置如下：

TQ_BRP = BRP - 1 = 5

PHSEG1 = T_{SEG1} -1 = 12

PHSEG2 = T_{SEG2} - 1 = 5

SJW = (0~3) = 0

SAM = T_{SYNC_SEG} = 1

3.3 常见波特率的参考配置（Fbase =24M）

Setting / Rate	TQ_BRP	PHSEG1	PHSEG2	SJW	SAM	通信距离
50Kbps	11	12	5	0	1	<=1.3KM
100Kbps	5	12	5	0	1	<=800M
125Kbps	5	10	3	1	1	<=500M
250Kbps	2	10	3	1	1	<=270M
500Kbps	0	15	6	1	1	<=100M
1Mbps	0	7	2	1	1	<=40M

4. CAN 初始化

4.1 GPIO 配置

两组 GPIO 可供配置：A3/A4，A9/A10。

```
typedef enum
{
    CAN_PORT_A3_A4 = 0,
    CAN_PORT_A9_A10,
}CAN_PORT_MODE;

CAN_PortSelect(CAN_PORT_A9_A10);
```

4.2 波特率配置

```
CAN_INIT_STRUCT can_init;
```

```
can_init.PHSEG1  = Can_BaudRate[baudrate][0];
can_init.PHSEG2  = Can_BaudRate[baudrate][1];
can_init.SAM     = Can_BaudRate[baudrate][2];
can_init.TQ_BRP = Can_BaudRate[baudrate][3];
can_init.SJW     = Can_BaudRate[baudrate][4];
//Acceptance code and mask
can_init.CAN_ACPC= 0x00;
can_init.CAN_ACPM= 0xffffffff;
CAN_Init(&can_init);
```

4.3 时钟配置

```
CAN_ClkSelect(CAN_CLK_OSC_24M);
```

4.4 初始化参考配置 API

can_interface.c 提供了 CAN_ModuleInit 接口，直接调用既可以完成参考配置。

```
CAN_ModuleInit(RATE_500KBPS,CAN_PORT_A3_A4);
```

5. CAN 中断

5.1 CAN 中断类型

- CAN_INT_RX_EN: 接收完成中断
- CAN_INT_TX_EN: 发送完成中断
- CAN_INT_ERR_WRN_EN: 错误/警告中断
- CAN_INT_OR_EN: 接收 FIFO 溢出中断
- CAN_INT_WAKEUP_EN: 唤醒中断
- CAN_INT_ERR_PASSIVE_EN: Error Passive Interrupt
- CAN_INT_ARB_LOST_EN: Arbitration Lost Interrupt
- CAN_INT_BERR_EN: Bus Error Interrupt

5.2 CAN 中断状态 CAN_BIT_INTSTATUS

```
CAN_BIT_INTSTATUS int_flag = CAN_GetIntStatus();
```

- CAN_INT_RX_FLAG: Receive Interrupt Flag
- CAN_INT_TX_FLAG: Transmit Interrupt Flag
- CAN_INT_ERR: Error Warning Interrupt
- CAN_INT_DATA_OR: Data Overrun Interrupt
- CAN_INT_WAKEUP: Wake-Up Interrupt
- CAN_INT_ERR_PASSIVE: Error Passive Interrupt
- CAN_INT_ARB_LOST: Arbitration Lost Interrupt
- CAN_INT_BERR: Bus Error Interrupt

5.3 CAN 中断配置

```
//配置 CAN 接收中断和接收 FIFO 溢出中断
CAN_IntTypeEnable(CAN_INT_OR_EN | CAN_INT_RX_EN);
//CAN 中断和 SPDIF 复用一个中断号 22
NVIC_EnableIRQ(SPDIF_IRQn);//22
GIE_ENABLE();
```

5.4 CAN 中断服务函数

```
//CAN 中断和 SPDIF 中断复用一个中断服务函数
void SPDIF0_Interrupt(void)
{
    CAN_DATA_MSG msg;
    CAN_BIT_INTSTATUS int_flag = CAN_GetIntStatus();

    if(int_flag & CAN_INT_RX_FLAG)
    {
        //接收数据
        CAN_RecvISR(&msg);
        CAN_ClrIntStatus(CAN_INT_RX_FLAG);
    }
    if(int_flag & CAN_INT_DATA_OR)
    {
        //数据溢出，接收 fifo 中数据然后 RST
        uint8_t cnt;
        cnt = CAN_GetRxMsgCnt();

        while(cnt--)
        {
            CAN_RecvISR(&msg);
        }
        CAN_SetModeCmd(CAN_MODE_RST_SELECT);
        CAN_SetModeCmd(CAN_MODE_RST_DISABLE);
    }
    if(int_flag & CAN_INT_WAKEUP)
    {
        //唤醒中断
        CAN_ClrIntStatus(CAN_INT_WAKEUP);
    }
    if(int_flag & CAN_INT_TX_FLAG)
    {
        //发送中断
        CAN_ClrIntStatus(CAN_INT_TX_FLAG);
    }
    if(int_flag & CAN_INT_BERR)
```

```
    {
        CAN_ClrIntStatus(CAN_INT_BERR);
    }
    if(int_flag & CAN_INT_ERR)
    {
        CAN_ClrIntStatus(CAN_INT_ERR);
    }
    if(int_flag & CAN_INT_ERR_PASSIVE)
    {
        CAN_ClrIntStatus(CAN_INT_ERR_PASSIVE);
    }
    if(int_flag & CAN_INT_ARB_LOST)
    {
        CAN_ClrIntStatus(CAN_INT_ARB_LOST);
    }
}
```

6. CAN 数据发送接收

6.1 CAN 状态 CAN_BIT_STATUS

```
CAN_BIT_STATUS CAN_GetStatus(void)
```

- CAN_RX_RDY：　　Rx Buffer Ready
  1 Rx buffer is not empty.
  0 Rx buffer is empty.

- CAN_DATA_OR_FLAG ：　Data overrun
  1 data buffer overrun
  0 data buffer not overrun

- CAN_TX_RDY：　　Tx Buffer Ready
  1 Tx buffer ready
  0 Tx buffer not ready

- CAN_TX_OVER:　　发送完成标志位
- CAN_RX_STA：　　Receive Status
- CAN_TX_STA：　　Transmit Status
- CAN_ERR_STA:　　Error Status

- CAN_BUS_STA:　　Bus Status
  1 - 总线关闭

6.2 CAN 数据发送

- 阻塞方式发送：

```
CAN_Send(&Msg,500);
```

- 轮询方式发送：

```
CAN_SendToBuf(&Msg);
CAN_SetModeCmd(CAN_CMD_TRANS_REQ);
while(!(CAN_GetStatus() & CAN_TX_OVER))
{
    //等待发送完成，这里可以干其他事
    DBG(".");
}
```

- 中断方式发送：

配置中断：`CAN_IntTypeEnable(CAN_INT_TX_EN);`

发送数据：

```
    CAN_SendToBuf(&Msg);
    CAN_SetModeCmd(CAN_CMD_TRANS_REQ);
```

发送完成：

```
    void SPDIF0_Interrupt(void)
    {
        CAN_BIT_INTSTATUS int_flag = CAN_GetIntStatus();
        if(int_flag & CAN_INT_TX_FLAG)
        {
            //发送中断
            CAN_ClrIntStatus(CAN_INT_TX_FLAG);
        }
    }
```

6.3 CAN 数据接收

- 阻塞方式接收：

```
CAN_Recv(&Msg,500);
```

- 轮询方式接收：

方式 1：检查 Rx buffer 不为空

```
    if(CAN_GetStatus() & CAN_RX_RDY)
    {
        CAN_RecvISR(&Msg);
    }
```

方式 2：检查 receive message 计数器不为空

```
    if(CAN_GetRxMsgCnt() > 0)
    {
        CAN_RecvISR(&Msg);
    }
```

- 中断方式接收：

```
CAN_RecvISR(&msg);
```

配置中断：`CAN_IntTypeEnable(CAN_INT_RX_EN);`

```
void SPDIF0_Interrupt(void)
{
    CAN_DATA_MSG msg;
    CAN_BIT_INTSTATUS int_flag = CAN_GetIntStatus();

    if(int_flag & CAN_INT_RX_FLAG)
    {
        //接收数据
        CAN_RecvISR(&msg);
        CAN_ClrIntStatus(CAN_INT_RX_FLAG);
    }
}
```

7. CAN 工作模式

7.1 Reset Mode

- CAN 模块复位：

```
CAN_SetModeCmd(CAN_MODE_RST_SELECT);
CAN_SetModeCmd(CAN_MODE_RST_DISABLE);
```

- 初始化：

```
CAN_SetModeCmd(CAN_MODE_RST_SELECT);

can_init.PHSEG1  = Can_BaudRate[baudrate][0];
can_init.PHSEG2  = Can_BaudRate[baudrate][1];
can_init.SAM     = Can_BaudRate[baudrate][2];
can_init.TQ_BRP    = Can_BaudRate[baudrate][3];
can_init.SJW   = Can_BaudRate[baudrate][4];
can_init.CAN_ACPC= 0x00;
can_init.CAN_ACPM= 0xffffffff;
CAN_Init(&can_init);

CAN_SetModeCmd(CAN_MODE_RST_DISABLE);
```

- 部分命令必须在该模式下操作：

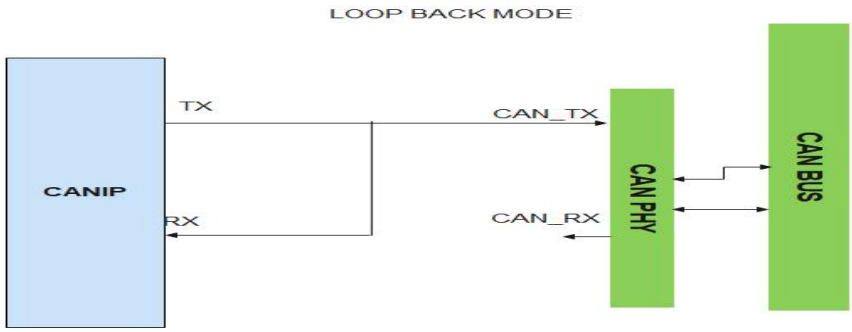
```
CAN_SetModeCmd(CAN_MODE_RST_SELECT);
CAN_SetModeCmd(CAN_MODE_AUWK_MODE);
CAN_SetModeCmd(CAN_MODE_SLEEP_SEL);
CAN_SetModeCmd(CAN_MODE_RST_DISABLE);
```

7.2 Test Mode

- Loopback mode

```
CAN_SetModeCmd(CAN_MODE_RST_SELECT);
CAN_SetModeCmd(CAN_MODE_LB_MOD);
CAN_SetModeCmd(CAN_MODE_RST_DISABLE);
```

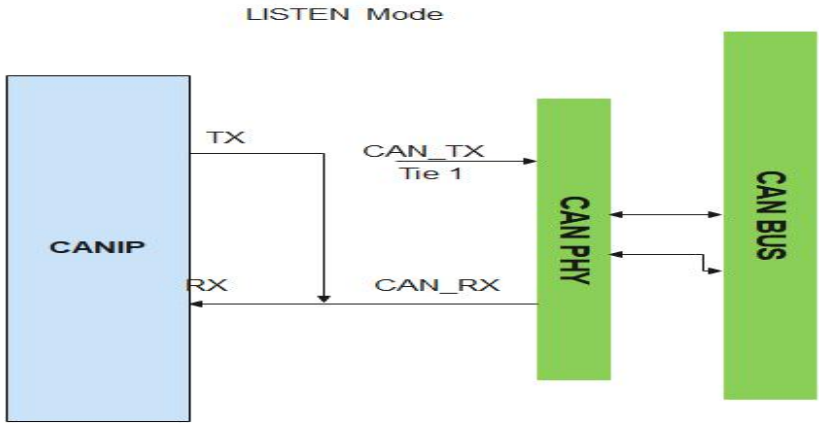
TX 发给 bus，RX 内部接自己的 TX，PHY 的 CAN RX 悬空。



- Listen mode（监听模式）

```
CAN_SetModeCmd(CAN_MODE_RST_SELECT);
CAN_SetModeCmd(CAN_MODE_LST_ONLY);
CAN_SetModeCmd(CAN_MODE_RST_DISABLE);
```

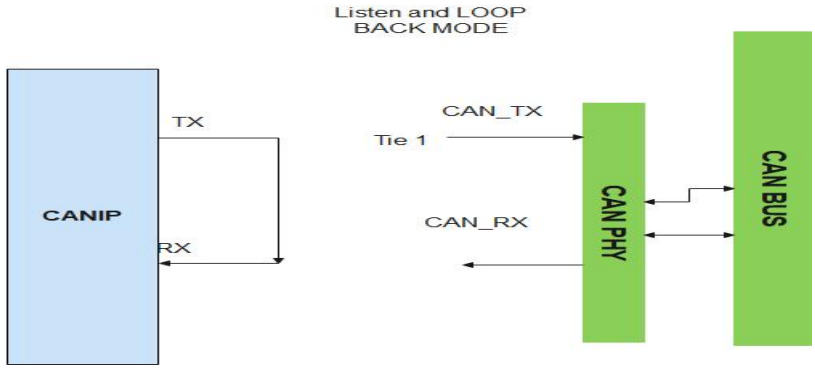
RX 接收来自 bus 信号，内部 TX 接到内部 RX 。



- Lissten and loopback mode（回环测试）

```
CAN_SetModeCmd(CAN_MODE_RST_SELECT);
CAN_SetModeCmd(CAN_MODE_LST_ONLY);
CAN_SetModeCmd(CAN_MODE_LB_MOD);
CAN_SetModeCmd(CAN_MODE_RST_DISABLE);
```

内部 tx 接内部 RX。



回环测试：

```
//测试数据发送
CAN_SendToBuf(&Msg);
CAN_SetModeCmd(CAN_CMD_SELF_REQ);
while(!(CAN_GetStatus() & CAN_TX_OVER))
{
    DBG(".");
}

//数据接收
if(CAN_GetRxMsgCnt() > 0)
{
    CAN_RecvISR(&Msg);
}
```

8. CAN 异常处理

8.1 Rx/Tx 错误计数器 CAN_RXTX_ERR_CNT

```
typedef struct
{
    uint8_t ERR_WRN_LMT;
    uint8_t RX_ERR_CNT;
    uint8_t TX_ERR_CNT;
}CAN_RXTX_ERR_CNT;
```

- 设置边界/清除错误计数器：

```
CAN_RXTX_ERR_CNT cnt;
cnt.ERR_WRN_LMT = 0x80;
cnt.RX_ERR_CNT  = 0;
cnt.TX_ERR_CNT  = 0;
CAN_SetRxTxErrCnt(&cnt);
```

- 读取 CAN 计数器：

```
CAN_RXTX_ERR_CNT cnt;
Cnt = CAN_GetRxTxErrCnt();
```

8.2 总线关闭

总线关闭以后，如何恢复通信：

```
if(CAN_GetStatus() & CAN_BUS_STA) //总线关闭
{
    //需要等总线到 128 次 11bit 隐性位后软件写 BUS_OFF_REQ
    DelayMs(10);
    CAN_SetModeCmd(CAN_CMD_BUS_OFF);
}
```

8.3 数据接收 FIFO 溢出

数据如果没有及时取走，FIFO 会发生溢出。溢出以后为了能正常使用 CAN，必须进行一次复位。

- 查询溢出状态位：

```
if(CAN_GetStatus() & CAN_DATA_OR_FLAG)
{
    //数据溢出，先接收 fifo 中的数据，然后 RST
    uint8_t cnt;
    cnt = CAN_GetRxMsgCnt();

    while(cnt--)
    {
        CAN_RecvISR(&msg);
    }
    CAN_SetModeCmd(CAN_MODE_RST_SELECT);
    CAN_SetModeCmd(CAN_MODE_RST_DISABLE);
}
```

- 如果开启了中断，会产生溢出中断：

```
if(CAN_GetIntStatus() & CAN_INT_DATA_OR)
{
    //数据溢出，先接收 fifo 中的数据，然后 RST
    uint8_t cnt;
    cnt = CAN_GetRxMsgCnt();

    while(cnt--)
    {
        CAN_RecvISR(&msg);
    }
    CAN_SetModeCmd(CAN_MODE_RST_SELECT);
    CAN_SetModeCmd(CAN_MODE_RST_DISABLE);
}
```