CAN自适应波特率

1. 需求：CAN总线上只有250K和500K两种波特率，需要在这两个波特率之间切换，以匹配总线上的数据传输速率
2. 解决思路：先以某个波特率初始化CAN控制器，CAN工作在普通模式。初始化后，CAN控制器不向外发送数据，只等待接收数据。如果Message buffer能够收到数据，说明当前的波特率是匹配的；反之，CAN控制器由于速率不匹配就会收到错误帧，检测到错误帧之后切换波特率，恢复正常通信。
3. 操作步骤：
   a. CAN先初始化为250K波特率，初始配置如下：

```
can_user_config_t can_pal_Config = {
    .maxBuffNum = 32UL,
    .mode = CAN_NORMAL_MODE,
    .peClkSrc = CAN_CLK_SOURCE_OSC,
    .enableFD = false,
    .payloadSize = CAN_PAYLOAD_SIZE_8,
#if (CAN_BAUD_250K == 1)
    .nominalBitrate = {
        .propSeg = 4UL,
        .phaseSeg1 = 5UL,
        .phaseSeg2 = 3UL,
        .preDivider = 5UL,
        .rJumpwidth = 3UL
    },
#else
    .nominalBitrate = {
        .propSeg = 4UL,
        .phaseSeg1 = 5UL,
        .phaseSeg2 = 3UL,
        .preDivider = 2UL,
        .rJumpwidth = 3UL
    },
#endif
    .extension = NULL
};
```

   b. 主函数程序如下所示：

```c
int main(void)
{
    uint8_t cnt = 0;
    status_t status = STATUS_SUCCESS;
    uint8_t canSentFrameCnt = 0;
    uint32_t initSysTime = 0;
    uint32_t currentSysTime = 0;
    bool flag_250K;
    bool flag_500K;
    /* Set information about the data to be sent
     *  - Standard message ID
     *  - Bit rate switch enabled to use a different bitrate for the data segment
     *  - Flexible data rate enabled
     *  - Use zeros for FD padding
     */
    const can_buff_config_t stdbuffCfg = {
        .enableFD = false,
        .enableBRS = true,
        .fdPadding = 0U,
        .idType = CAN_MSG_ID_STD,
        .isRemote = false};
    const can_buff_config_t rxExtBuffCfg = {
        .enableFD = false,
        .enableBRS = true,
        .fdPadding = 0U,
        .idType = CAN_MSG_ID_EXT,
        .isRemote = false};
    /* Set information about the data to be sent
     *  - Standard message ID
     *  - Bit rate switch enabled to use a different bitrate for the data segment
     *  - Flexible data rate enabled
     *  - Use zeros for FD padding
     */
    /* Initialize clocks*/
    OSIF_TimeDelay(1);
    status |= CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                             g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
    status |= CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_AGREEMENT);
    if (status != STATUS_SUCCESS)
    {
        return STATUS_ERROR;
    }

    /* Initialize pins */
    status |= PINS_DRV_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);
    /* Initialize UART instance */
    status |= UART_DRV_Init(PRINTF_UART, &UartState, &UartInitConfig);
    status |= CAN_Init(&can_pal_instance, &can_pal_config);
    initSysTime = OSIF_GetMilliseconds();
    status |= CAN_ConfigRxBuff(&can_pal_instance, RX_MAILBOX, &stdbuffCfg, RX_MSG_ID);
    status |= CAN_ConfigRxBuff(&can_pal_instance, RX_EXT_MAILBOX, &rxExtBuffCfg, RX_EXTMSG_ID);
    status |= CAN_ConfigTxBuff(&can_pal_instance, TX_MAILBOX, &stdbuffCfg);

    /* Start receiving data in RX_MAILBOX. */
    status |= CAN_Receive(&can_pal_instance, RX_MAILBOX, &rxMsg);
    status |= CAN_Receive(&can_pal_instance, RX_EXT_MAILBOX, &rxExtMsg);
```

```c
while (1)
{
    if (CAN_GetTransferStatus(&can_pal_instance, RX_MAILBOX) != STATUS_BUSY)
    {
        /* Start receiving data in RX_MAILBOX. */
        status |= CAN_Receive(&can_pal_instance, RX_MAILBOX, &rxMsg);
    }
    if (CAN_GetTransferStatus(&can_pal_instance, RX_EXT_MAILBOX) != STATUS_BUSY)
    {
        /* Start receiving data in RX_MAILBOX. */
        status |= CAN_Receive(&can_pal_instance, RX_EXT_MAILBOX, &rxExtMsg);
    }
    // If an error occurs, exit the loop
    if ((CAN0->ESR1 & CAN_ESR1_ERRINT_MASK) >> CAN_ESR1_ERRINT_SHIFT == 1)
    {
        OSIF_TimeDelay(10);
        CAN0->ESR1 = CAN_ESR1_ERRINT_MASK;
        flag_500K = true;
        break;
    }

    currentSysTime = OSIF_GetMilliseconds();
    if (currentSysTime - initSysTime >= DETECT_TIME)
    {
        if (((rxMsg.id == 0x02) || (rxExtMsg.id == 0x70004)))
        {
            flag_250K = true;
            break;
        }
    }
    OSIF_TimeDelay(100);
}
```

初始化之后开启标准帧和扩展帧的接收，并同时通过ESR1寄存器检测是否出现错误帧。若出现错误帧，则清除错误标志，再将flag_500K设为true，表示当前的250K波特率与总线速率不匹配；若3S内都没有出现错误帧，则对接收缓冲区接收到的数据ID进行判断，如果与正确的ID保持一致，说明通讯成功，当前的波特率有效，flag_250K设为true。

c. 当第一个循环中判定完成之后，即可根据波特率标志来执行相应的操作。

```c
if (flag_250K)
{
    while (1)
    {
        /* Send the information via CAN */
        if (CAN_GetTransferStatus(&can_pal_instance, TX_MAILBOX) != STATUS_BUSY)
        {
            status |= CAN_Send(&can_pal_instance, TX_MAILBOX, &txMsg);
            canSentFrameCnt++;
            PRINTF("DATA %d sent\n", txMsg.data[0]);
        }
        if (CAN_GetTransferStatus(&can_pal_instance, RX_MAILBOX) != STATUS_BUSY)
        {
            /* Start receiving data in RX_MAILBOX. */
            status |= CAN_Receive(&can_pal_instance, RX_MAILBOX, &rxMsg);
        }
        if (CAN_GetTransferStatus(&can_pal_instance, RX_EXT_MAILBOX) != STATUS_BUSY)
        {
            /* Start receiving data in RX_MAILBOX. */
            status |= CAN_Receive(&can_pal_instance, RX_EXT_MAILBOX, &rxExtMsg);
        }
        OSIF_TimeDelay(100);
        if (canSentFrameCnt == 100)
        {
            break;
        }
    }
}
```

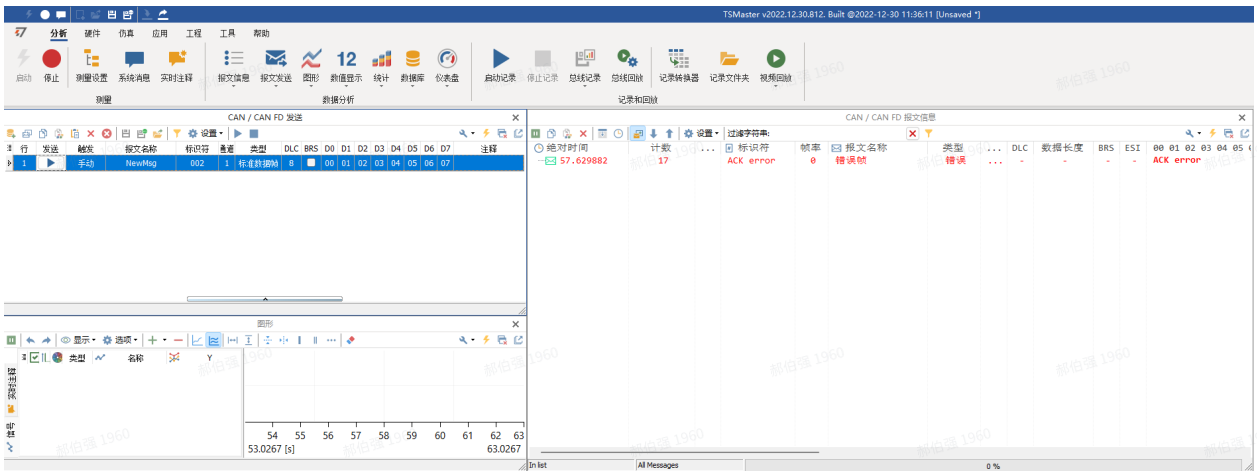
```c
else if (flag_500K)
{
    FLEXCAN_DRV_SetBitrate(can_pal_instance.instIdx, &CanBaud500K);

    while (1)
    {
        txMsg.data[0]++;
        cnt++;
        /* Send the information via CAN */
        if (CAN_GetTransferStatus(&can_pal_instance, TX_MAILBOX) != STATUS_BUSY)
        {
            status |= CAN_Send(&can_pal_instance, TX_MAILBOX, &txMsg);
            canSentFrameCnt++;
            PRINTF("DATA %d sent\n", txMsg.data[0]);
        }
        if (CAN_GetTransferStatus(&can_pal_instance, RX_MAILBOX) != STATUS_BUSY)
        {
            PRINTF("Received MSG ID = %x, data = %x\n", rxMsg.id, rxMsg.data[0]);
            /* Start receiving data in RX_MAILBOX. */
            status |= CAN_Receive(&can_pal_instance, RX_MAILBOX, &rxMsg);
        }
        if (CAN_GetTransferStatus(&can_pal_instance, RX_EXT_MAILBOX) != STATUS_BUSY)
        {
            PRINTF("Received EXT MSG ID = %x, data = %x\n", rxExtMsg.id, rxExtMsg.data[0]);
            /* Start receiving data in RX_MAILBOX. */
            status |= CAN_Receive(&can_pal_instance, RX_EXT_MAILBOX, &rxExtMsg);
        }
        OSIF_TimeDelay(100);
        if (canSentFrameCnt == 100)
        {
            break;
        }
    }
}
else
{
    return STATUS_ERROR;
}
```

如果flag_250K=true，则可以直接以当前配置向CAN总线发送数据进行通讯；如果flag_500K=true，则需要通过函数**FLEXCAN_DRV_SetBitrate(can_pal_instance.instIdx, &CanBaud500K)**重新设置波特率为500K。该波特率的时间段配置如下：

```
flexcan_time_segment_t CanBaud500K = {
    .propSeg = 4,
    .phaseSeg1 = 5,
    .phaseSeg2 = 3,
    .preDivider = 2,
    .rJumpwidth = 3,
};
```

4. 测试结果

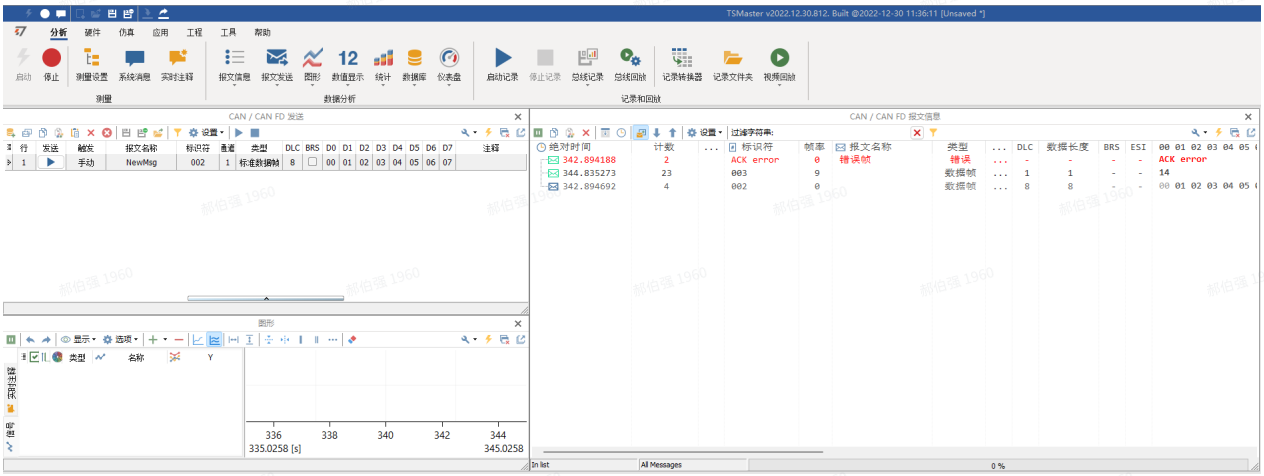
上位机以500K速率发来数据时，CAN控制器接收到错误帧



```
102     status |= CAN_Init(&can_pal_instance, &can_pal_Config);
103     initSysTime = OSIF_GetMilliseconds();
104     status |= CAN_ConfigRxBuff(&can_pal_instance, RX_MAILBOX, &stdbuffCfg, RX_MSG_ID);
105     status |= CAN_ConfigRxBuff(&can_pal_instance, RX_EXT_MAILBOX, &rxExtBuffCfg, RX_EXTMSG_ID);
106     status |= CAN_ConfigTxBuff(&can_pal_instance, TX_MAILBOX, &stdbuffCfg);
107
108     /* Start receiving data in RX_MAILBOX. */
109     status |= CAN_Receive(&can_pal_instance, RX_MAILBOX, &rxMsg);
110     status |= CAN_Receive(&can_pal_instance, RX_EXT_MAILBOX, &rxExtMsg);
111
112     while (1)
113     {
114         if (CAN_GetTransferStatus(&can_pal_instance, RX_MAILBOX) != STATUS_BUSY)
115         {
116             /* Start receiving data in RX_MAILBOX. */
117             status |= CAN_Receive(&can_pal_instance, RX_MAILBOX, &rxMsg);
118         }
119         if (CAN_GetTransferStatus(&can_pal_instance, RX_EXT_MAILBOX) != STATUS_BUSY)
120         {
121             /* Start receiving data in RX_MAILBOX. */
122             status |= CAN_Receive(&can_pal_instance, RX_EXT_MAILBOX, &rxExtMsg);
123         }
124         // If an error occurs, exit the loop
125         if ((CAN0->ESR1 & CAN_ESR1_ERRINT_MASK) >> CAN_ESR1_ERRINT_SHIFT == 1)
126         {
127             OSIF_TimeDelay(10);
128             CAN0->ESR1 = CAN_ESR1_ERRINT_MASK;
129             flag_500K = true;
130             break;
131         }
132
133         currentSysTime = OSIF_GetMilliseconds();
134         if (currentSysTime - initSysTime >= DETECT_TIME)
135         {
136             if (((rxMsg.id == 0x02) || (rxExtMsg.id == 0x70004)))
137             {
138                 flag_250K = true;
139                 break;
140             }
141         }
142         OSIF_TimeDelay(100);
143     }
144     if (flag_250K)
145     {
146         while (1)
147         {
148             /* Send the information via CAN */
```

程序继续运行过后，波特率切换成功，恢复正常通讯

CAN0	642 Registers	FlexCAN module
MCR	0002 101F	CAN_MCR defines global system configurations,
CTRL1	02EB 0004	The CAN_CTRL1 register is defined for specifi
PRESDIV	2	The Sclock period defines the time quantum of
RJW	3	One time quantum is equal to the Sclock perio
PSEG1	5	The valid programmable values are 0â7. PSEG1
PSEG2	3	The valid programmable values are 1â7. PSEG2
BOFFMSK	0 (0)	
ERRMSK	0 (0)	
CLKSRC	0 (0)	The selected clock is fed to the prescaler to
LPB	0 (0)	In Loop-Back mode, the FlexCAN performs an in
TWRNMSK	0 (0)	TWRNMSK is read as zero when CAN_MCR.WRNEN is
RWRNMSK	0 (0)	RWRNMSK is read as zero when CAN_MCR.WRNEN is
SMP	0 (0)	SMP can be written only in Freeze mode becaus
BOFFREC	0 (0)	If BOFFREC is negated, automatic recovering f
TSYN	0 (0)	The Timer Sync feature provides a means to sy
LBUF	0 (0)	When LBUF is asserted, the CAN_MCR.LPRIOEN bi
LOM	0 (0)	In Listen-Only mode, transmission is disabled
PROPSEG	4	The valid programmable values are 0â7. PROPSE
TIMER	0000 B5D1	The CAN_TIMER register represents a 16-bit fr
RXMGMASK	FFFF FFFF	
RX14MASK	FFFF FFFF	The CAN_RX14MASK register is located in RAM.
RX15MASK	FFFF FFFF	The CAN_RX15MASK register is located in RAM.



5. 总结

检测到错误帧时ESR1[ERRINT]置1就表示BIT1ERR，BIT0ERR，ACKERR，CRCERR，FRMERR或者 STFERR其中一种或多种错误发生了。ESR1[ERRINT]置1后不能马上清标志，否则会进入错误中断，程序卡死，需要等待几毫秒，等所有错误标志位稳定之后，再清标志。