ACM32F403/FP401/A403/F0X0/A070/WB15
CAN 应用笔记

目录

前言

CAN 是控制器局域网络(Controller Area Network)的简称，它是由研发和生产汽车电子产品著称的德国 BOSCH 公司开发的，并最终成为国际标准（ISO11519），是国际上应用最广泛的现场总线之一。

CAN 总线协议已经成为汽车计算机控制系统和嵌入式工业控制局域网的标准总线，并且拥有以 CAN 为底层协议专为大型货车和重工机械车辆设计的 J1939 协议。近年来，它具有的高可靠性和良好的错误检测能力受到重视，被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。

本应用笔记举例介绍了 CAN 外设的相关同能，以及距离说明了如何进行 CAN 通讯。本应用笔记主要包括两部分内容：

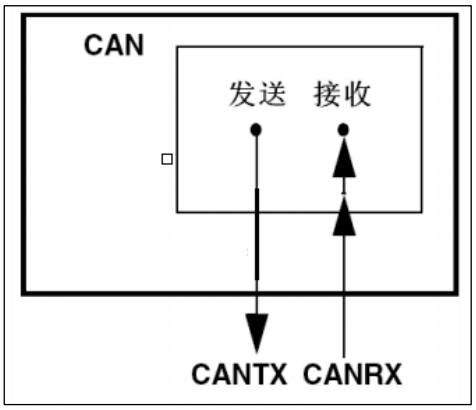
- 第 1 部分介绍 CAN 模块的主要特性。
- 第 2 部分介绍 CAN 用于报文收发通讯。

一、 CAN 主要特性

1.1 工作模式

1.1.1 Operating Mode

可以正常发送接收 CAN 总线数据，进入 Operating Mode 的方式是清除 MOD 的 BIT0。

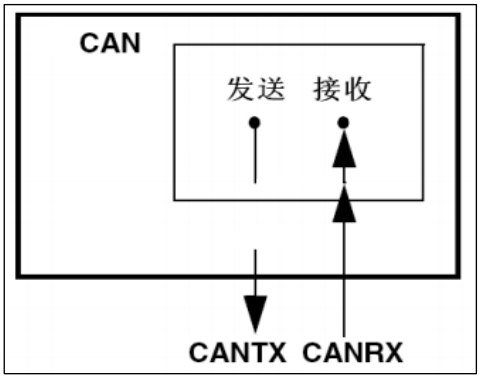


1.1.2 Reset Mode

可以修改时间参数和报文过滤参数，进入 Reset Mode 的方式有两种：第一种就是执行一次硬件复位，第二种方式是写 MOD 寄存器的 BIT0。

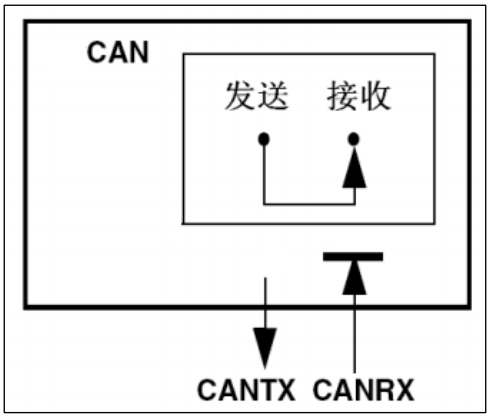
1.1.3 监听模式

CAN 可以支持监听模式，在 Reset Mode 和 Operating Mode 两种模式下都可使能。在监听模式下，CAN 只能用于接收数据，不能发送。CAN 在成功接收到一帧数据时也不会回复 ACK。并且会强制进入“Error Passive”模式。这种模式允许软驱动波特率检测，使 CAN 支持 hot plug-in 模式,能够自适应波特率。



1.1.4 自测试模式

CAN 可以支持自测模式，在 Reset Mode 和 Operating Mode 两种模式下都可使能。自测试模式下，CAN 通过自己接收的模式进行发送和接收，不需要从远程节点获取 ACK。使用自测试模式可以进行自检，而不需要外接任何 CAN 节点。



1.1.5 睡眠模式

如果 Bus 是空闲的，并且没有中断被挂起，那么可以将 CAN 模块进入那 Sleep 模式。进入 Sleep 模式的方式是置位 MOD.4。TX0 在 Sleep 模式时为高。

可以通过以下方式唤醒：

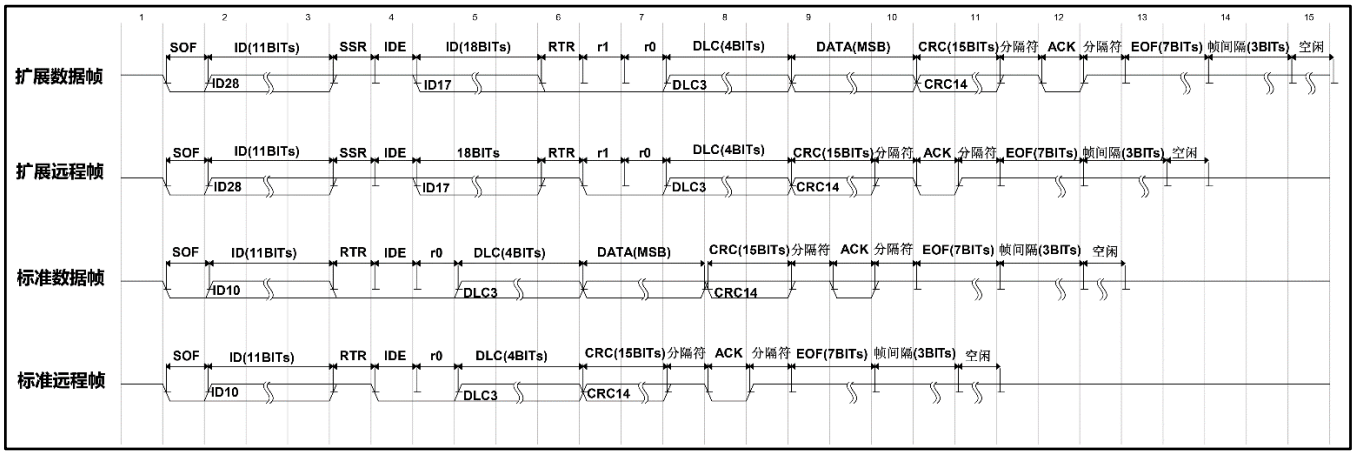
1. 将 Sleep 模式设为 0。
2. 在 RX0 上检查的数据。
3. NINT_IN 有一个低电平。
4. 唤醒后，CAN 会产生一个 Wake-Up 中断

1.1.6 时钟输出模式

CAN 提供了一种时钟输出模式，但是只能在 Reset Mode 时需选择，TX0 用来输出发送的时钟而不是数据。使用时需要通过设置 OCR 寄存器，设置 TX 引脚功能，如作为 TX 发送引脚，或者输出 Tx_clock，或输出 Clock out。作为 Clock out 时，还需要在 CDR 寄存器中对输出的 PCLK 时钟进行设置。

1.2 发送

根据 CAN 数据帧的结构，如图所示。将需要发送的帧类型、帧长度、帧数据写入到 CAN 的 TxBuff 中，不管数据帧格式是标准帧还是扩展帧，TxBuff 由 13 字节的数据组成，从地址 0x10 到 0x1C，这样可以保证能写入一个数据长度为 8 字节的数据帧。注：在向 TxBuff 写数据之前需要先检查一下 SR.2 状态，确保 TxBuff 被释放了，否则写入的数据就丢失。



为了将存放在 TxBuff 里面的数据发送出去，可以通过设置 CMR.0（发送请求）或者设置自接收请求 CMR.4。开始发送时，SR.5 被置为 1 并且发送请求被清除。

发送的比特流是通过 TX0 发送出去的，如果遇到仲裁失败或者发送错误，CAN 能够自动重发。

在每个数据帧后都会自动发送一个 15bit 的 CRC 校验值，而 CRC 是根据 SOF、仲裁域、控制域和数据域产生。

如果发送还没开始的话，可以通过写 CMR.1 中断一帧数据的发送，但是一旦开始了，就不能中断发送过程了。

TxBuff 的数据结构如图所示，由 13 个字节组成，发送时只需按照 TxBuff 结构填入相应的 RTR（远程帧 1/数据帧 0）、DLC（数据长度）、FF（标准帧 0/扩展帧 1）、ID 以及数据即可。

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
CAN Address	Field	CAN Address	Field
10h	TX Frame Information	10h	TX Frame Information
11h	TX Identifier 1	11h	TX Identifier 1
12h	TX Identifier 2	12h	TX Identifier 2
13h	TX Data Byte 1	13h	TX Identifier 3
14h	TX Data Byte 2	14h	TX Identifier 4
15h	TX Data Byte 3	15h	TX Data Byte 1
16h	TX Data Byte 4	16h	TX Data Byte 2
17h	TX Data Byte 5	17h	TX Data Byte 3
18h	TX Data Byte 6	18h	TX Data Byte 4
19h	TX Data Byte 7	19h	TX Data Byte 5
1Ah	TX Data Byte 8	1Ah	TX Data Byte 6
1Bh	(Unused)	1Bh	TX Data Byte 7
1Ch	(Unused)	1Ch	TX Data Byte 8

Transmit Frame (SFF)

CAN Address	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
10h	FF	RTR	X (1)	X (1)	DLC.3	DLC.2	DLC.1	DLC.0
11h	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21
12h	ID.20	ID.19	ID.18	X (2)	X (1)	X (1)	X (1)	X (1)

Transmit Frame (EFF)

CAN Address	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
10h	FF	RTR	X (1)	X (1)	DLC.3	DLC.2	DLC.1	DLC.0
11h	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21
12h	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13
13h	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5
14h	ID.4	ID.3	ID.2	ID.1	ID.0	X (2)	X (1)	X (1)

1.3 接收

CAN 接收的数据首先通过接收滤波器才能写入到 RxBuff。接收滤波器只会通过那些标识符符合条件的报文。

数据放到 RxBuff 时，SR.4 寄存器就会被置位。一旦接收到数据，SR.0 就会被置位，同时才生一个接收中断。RxBuff 的数据结构如图所示，由 13 个字节组成，接收时只需按照 RxBuff 结构读取相应的 RTR（远程帧 1/数据帧 0）、DLC（数据长度）、FF（标准帧 0/扩展帧 1）、

ID 以及数据即可。

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
CAN Address	Field	CAN Address	Field
10h	RX Frame Information	10h	RX Frame Information
11h	RX Identifier 1	11h	RX Identifier 1
12h	RX Identifier 2	12h	RX Identifier 2
13h	RX Data Byte 1	13h	RX Identifier 3
14h	RX Data Byte 2	14h	RX Identifier 4
15h	RX Data Byte 3	15h	RX Data Byte 1
16h	RX Data Byte 4	16h	RX Data Byte 2
17h	RX Data Byte 5	17h	RX Data Byte 3
18h	RX Data Byte 6	18h	RX Data Byte 4
19h	RX Data Byte 7	19h	RX Data Byte 5
1Ah	RX Data Byte 8	1Ah	RX Data Byte 6
1Bh	(Unused)	1Bh	RX Data Byte 7
1Ch	(Unused)	1Ch	RX Data Byte 8

Receive Frame (SFF)

CAN Address	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
10h	FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0
11h	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21
12h	ID.20	ID.19	ID.18	RTR	0	0	0	0

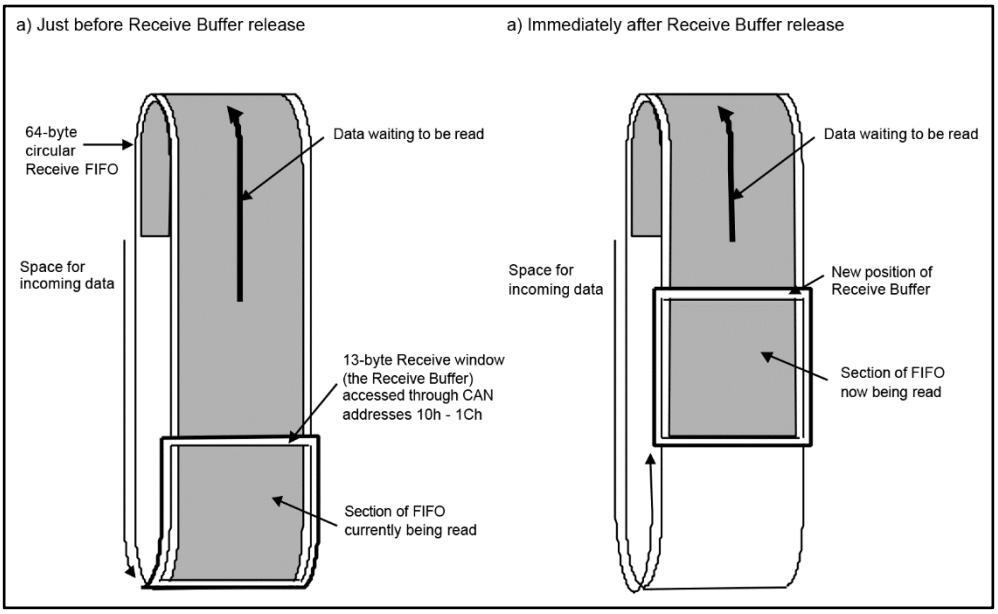
Receive Frame (EFF)

CAN Address	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
10h	FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0
11h	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21
12h	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13
13h	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5
14h	ID.4	ID.3	ID.2	ID.1	ID.0	RTR	0	0

RxFIFO 是 64 字节深度，最多允许存放 5 组 EFF 数据。如果数据没有被读取，RxFIFO 中没有空间接收新的数据，那么新的数据进来的话就会触发一次 Over Run，SR.1 会被置位，接收的数据也会被丢弃，也产生一个中断。

放在 RxBuff 中的数据通过一个 13 位宽的窗口读取，为了读取所有的报文，CPU 需要移动窗口，这是通过设置 CMR.2 来实现的。如果还有数据需要被读取，那么这个数据将会被移动到窗口，如果没有数据被读取了，那么接收标志就会自动清零，接收过程如图所示。

图 一-1 CAN 接收示意图



1.4 自我接收

CAN 可以接收自己发送给其他节点的数据。通过 CMR.4 来使能此功能。CAN 自动产生发送和接收中断。

此功能的作用在于让 CAN 总线可以直接同时接收和发送，而不需要其他节点配合，方便测试。

1.5 接收过滤

在 CAN 总线中,所有节点接收总线上的所有报文。为了让节点忽略与它无关的报文信息，CAN 允许对接收的报文进行过滤，通过一个 4 字节的接收过滤器实现。只有报文的标识符与过滤器匹配，才能被写入到 RxBuff。

CAN 一共有两种过滤模式：

- 单过滤模式：将 4 个 ACR 和 4 个 AMR 作为一个 32bit 的过滤器，过滤一组 ID 和数据，根据标准帧和扩展帧 ID 长度不同，单过滤模式过滤器结构也不同，结构如图所示。

Standard Frame Format, Single Filter

Receive Buffer: Address 11h *12h* *13h* *14h*

ID.28 .... ID.21	ID.20 ... ID18	RTR	X X X X *(not matched)*	Data Byte 1	Data Byte 2

Filter:

ACR0[7:0]	ACR1[7:4]	*(ACR1[3:0] unused)*	ACR2[7:0]	ACR3[7:0]
AMR0[7:0]	AMR1[7:4]	*(AMR1[3:0] unused)*	AMR2[7:0]	AMR3[7:0]

Note: If matching of the data bytes is not required, AMR2 and AMR3 should be set to FFh.

Extended Frame Format, Single Filter

Receive Buffer: Address 11h *12h* *13h* *14h*

ID.28 .... ID.21	ID.20... ID.13	ID.12 ... ID.5	ID.4 ... ID0	RTR	X X *(not matched)*

Filter:

ACR0[7:0]	ACR1[7:0]	ACR2[7:0]	ACR3[7:2]	*(ACR3[1:0] unused)*
AMR0[7:0]	AMR1[7:0]	AMR2[7:0]	AMR3[7:2]	*(AMR3[1:0] unused)*

- 双过滤模式：将 4 个 ACR 和 4 个 AMR 分别作为两个 16bit 的过滤器，过滤两组不同格式的 ID 和数据。如果使能 2 个过滤器，那么接收的报文只要符合其中一个条件，就可以被接收进来，双过滤模式过滤器结构如图所示。

Standard Frame Format, Dual Filters

Receive Buffer: Address 11h *12h* *13h* *14h*

ID.28 .... ID.21	ID.20 ... ID18	RTR	X X X X *(not matched)*	Data Byte 1 [7:4]	Data Byte 1 [3:0]	Data Byte 2 *(not matched)*

Filter 1:

ACR0[7:0]	ACR1[7:4]
AMR0[7:0]	AMR1[7:4]

ACR1[3:0]	ACR3[3:0]
AMR1[3:0]	AMR3[3:0]

Filter 2:

ACR2[7:0]	ACR3[7:4]
AMR2[7:0]	AMR3[7:4]

Extended Frame Format, Dual Filters

Receive Buffer: Address 11h *12h* *13h* *14h*

ID.28 .... ID.21	ID.20... ID.13	ID.12 ... ID.5 *(not matched)*	ID.4 ... ID0 *(not matched)*	RTR *(not matched)*	X X *(not matched)*

Filter 1:

ACR0[7:0]	ACR1[7:0]
AMR0[7:0]	AMR1[7:0]

Filter 2:

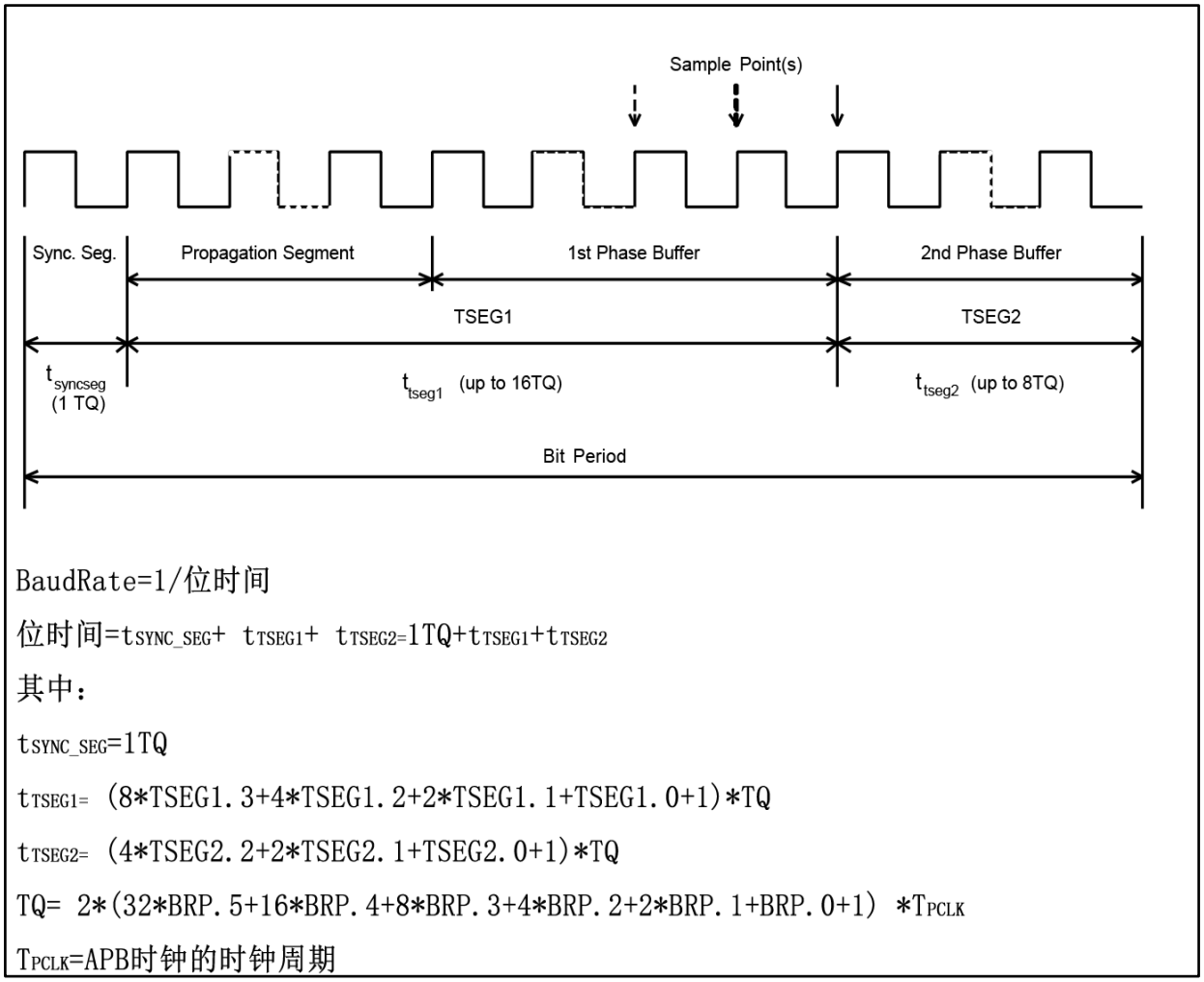
ACR2[7:0]	ACR3[7:0]
AMR2[7:0]	AMR3[7:0]

可以通过设置 AMRx 里相应的位来决定是否过滤 ACRx 中对应的位，当 AMRx 中的位为 1 时，表示“Don’t Care”，不对 ACRx 中相应位进行过滤；当 AMRx 中的位为 0 时，表示对 ACRx 中相应位进行过滤，只有当接收到的 CAN 报文 ID 对应位与 ACRx 中对应位一致，报文才能被接收进来写入到 RxBuff。

1.6 波特率

CAN 总线的波特率是通过每一个数据位的时序分解，分解的最小时间单位是 TQ，根据 BOSCH 标准，一个完整的位通常由 8-25 个 TQ 组成。

CAN 模块定义的位时序如图所示，共有三部分组成：



BaudRate=1/位时间

位时间=t_{SYNC_SEG}+ t_{TSEG1}+ t_{TSEG2}=1TQ+t_{TSEG1}+t_{TSEG2}

其中：

t_{SYNC_SEG}=1TQ

t_{TSEG1}= (8*TSEG1.3+4*TSEG1.2+2*TSEG1.1+TSEG1.0+1)*TQ

t_{TSEG2}= (4*TSEG2.2+2*TSEG2.1+TSEG2.0+1)*TQ

TQ= 2*(32*BRP.5+16*BRP.4+8*BRP.3+4*BRP.2+2*BRP.1+BRP.0+1) *T_{PCLK}

T_{PCLK}=APB时钟的时钟周期

- 同步段(SYNC_SEG)：通常期望位的变化发生在该时间段内，其值固定为 1TQ。
- 时间段 1(TSEG1)：定义采样点的位置。它包含 CAN 标准里的 PROP_SEG 和 PHASE_SEG1。 其值可以编程为 1 到 16 个 TQ。

- 时间段 2(TSEG2)：定义发送点的位置。它代表 CAN 标准里的 PHASE_SEG2。其值可以编程为 1 到 8 个 TQ。

重新同步跳跃宽度(SJW)定义了在每位中可以延长或缩短多少个时间单元的上限，其值可以为 1 到 4 个 TQ。

如设置波特率为 500K，系统时钟 64MHz，APB 时钟为 64MHz，则 BRP=7，TQ=2*(7+1)*(1/64M)=16*(1/64M)= 1/4M，设置 TSEG1=2，TSEG2=3，则数据位时间=1+(2+1)+(3+1)=8TQ，则 baud=1/8TQ=1/(8*1/4M)= 1/(2M)=500Kbps。

1.7 仲裁器

CAN 总线是由小的识别符节点控制的，失去仲裁的节点必须在总线空闲之前放弃控制总线。如果 CAN 仲裁失败，那么一个仲裁失败的中断将会起来，并且会记录失去仲裁的位置。

1.8 错误处理

CAN 包括两个错误计数器，接收错误计数器 RXERR 和发送错误接收器 TXERR。并且错误的类型和错误的位置都可以在 Error-Code-Capture-Register 中看到。CAN 也包括一个 EWL 寄存器，其值表示接收或者发送的错误值达到多少时产生一个警告。默认的 EWL 值为 96，不管接收错误还是发送错误达到这个值都会产生一个错误警告中断。

如果错误计数超过 127，那么 CAN 就会进入“Error Passive”状态。如果发送错误计数器超过 255，那么 SR.7 就会被置 1（Bus Off）。CAN 就会进入 Reset Mode 并且会产生一个 EI 中断。在重现进入 Bus On 状态前，CAN 必须等待 128 个 Bus-Free-Sequence。

1.9 Bus OFF 恢复

CAN 通过软件方式实现了 Bus OFF 恢复，使用时需要在初始化结构体中使能 CAN_ABOM，如图：

```
  uint32_t CAN_ABOM ;          /*!< CAN BUF OFF ERROR RECOVER
                               @ref CAN_ABOM e.g:CAN_ABOM_ENABLE  CAN_ABOM_DISABLE*/
} CAN_InitTypeDef;
```

当进入 Bus OFF 时，CAN 模块会进入复位模式，在中断处理函数中检测当前 CAN 模块

状态是否处于复位模式，若是则尝试恢复为正常模式，并等待 BUS OFF 恢复，总线处于激活状态即可。

```
void HAL_CAN_IRQHandler(CAN_HandleTypeDef *hcan)
{
    volatile uint32_t lu32_IR;
    lu32_IR = hcan->Instance->IR;//read clear

    if(lu32_IR & CAN_IR_RI) //RI
    {
        /* CAN ReceiveIT complete callback */
        HAL_CAN_GetRxMessage(hcan, hcan->RxMessage);
        hcan->CAN_ReceiveIT_Callback(hcan);
    }

    if(lu32_IR & CAN_IR_TI) //TI
    {
        /* CAN TransmitIT complete callback */
        hcan->CAN_TransmitIT_Callback(hcan);
    }

    if(hcan->Instance->MOD & CAN_MOD_RM) //bus off
    {
        /* CAN bus off error recover */
        HAL_CAN_OperatingModeRequest(hcan,CAN_OperatingMode_Normal);//enter CAN_OperatingMode_Normal
        while(hcan->Instance->SR& CAN_SR_BS);//wait bus off recover
    }
}
```

二、 CAN 收发示例

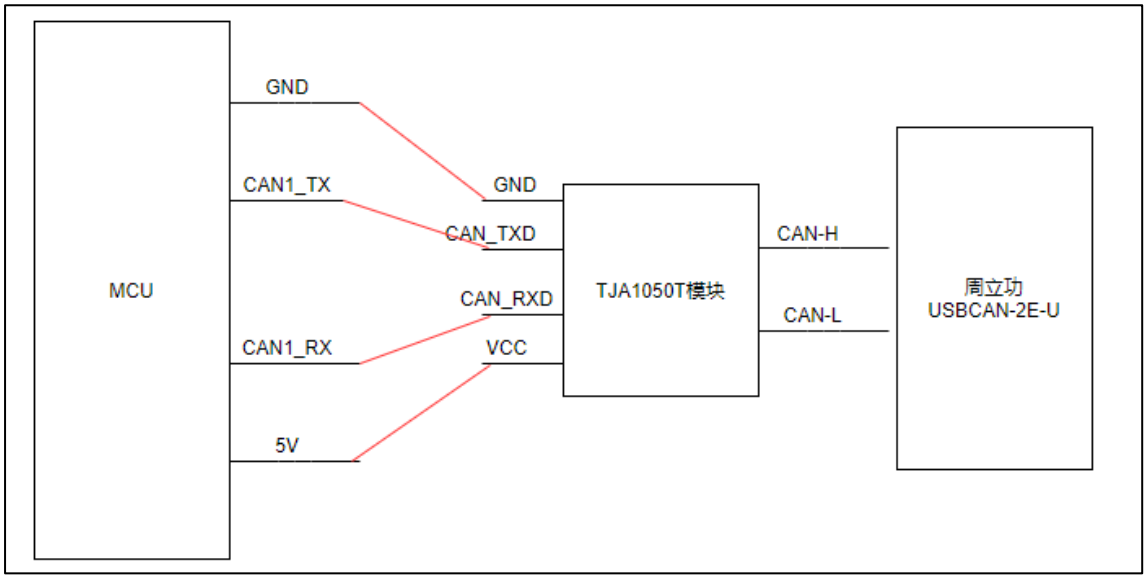
2.1 说明

本示例中说明了如何使用航芯的 ACM32xxx 芯片进行 CAN 报文收发通讯，例程中所用的 CAN 收发模块和 USB CAN 卡工具可会根据实际情况选用，使用方法类似。其中 USB CAN 卡工具仅为监控分析报文使用，若直接接入到整车 CAN 网络中，则无需此设备。

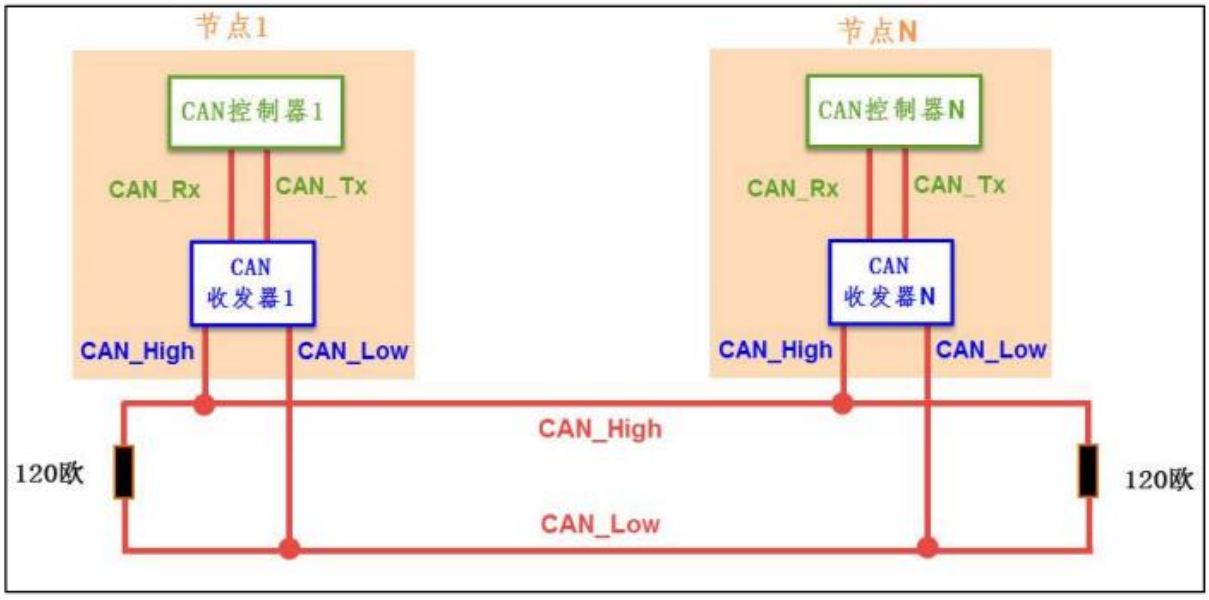
开发板	航芯 ACM32xxx 开发板
CAN 收发模块	TJA1050T 模块
USB CAN 卡工具	周立功 USBCAN-2E-U
监控软件	周立功 CANTest v2.68

2.2 硬件连接

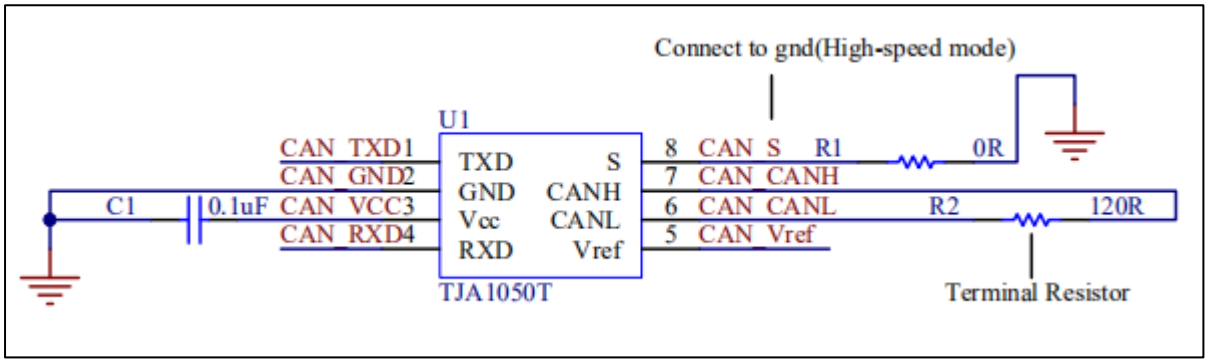
本示例中采用 ACM32xxx 的 CAN1 模块与周立功“USBCAN-2E-U”进行通讯，将 CAN1 的 TX 和 RX 与 CAN 收发“TJA1050T”模块的收发引脚相连接，注意 TJA1050T 模块供电为 5V。经 CAN 收发器转换后，将 CAN-H 和 CAN-L 连接到 CAN 总线网络中，本实例中与周立功 USBCAN-2E-U 相连，即可通过周立功的“CANTest”软件进行监控以及数据收发，其他 USB CAN 卡工具同理。



根据 CAN 闭环通讯通讯网络 ISO11898 标准，它的总线最大长度为 40m，通信速度最高为 1Mbps，总线的两端各要求有一个“120 欧”的终端电阻，如图所示。



其中，在本示例使用的 CAN 收发“TJA1050T”模块已经自带一个 120 欧的终端电阻，如图所示。若单独使用此芯片或使用其他 CAN 收发芯片时，需注意添加一个 120 欧的电阻。另外需要将周立功“USBCAN-2E-U”的 120 欧电阻开关打开，即实现了闭环网络所需的两个终端电阻要求。

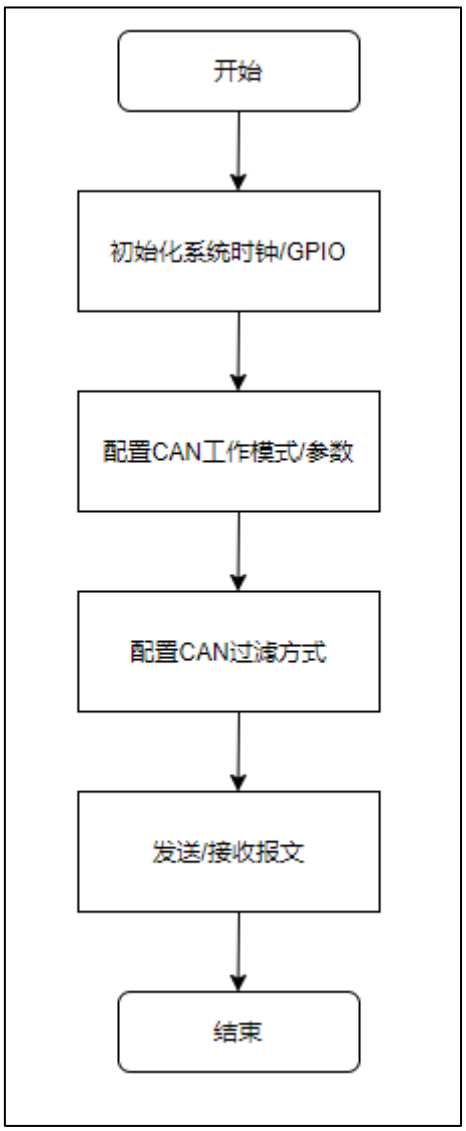


注：本实例中的 CAN 收发器芯片仅为参考，实际运用中可根据项目需求选择合适的收发器；终端电阻亦需要根据实际整车 CAN 网络情况，决定本 CAN 节点是否需要添加终端电阻，本示例中仅为 ACM32xxx 的 CAN 与周立功 USBCAN 卡两个节点进行通讯，因此电阻不可省略。

2.3 软件实现

2.3.1 软件流程

完成了上述的硬件连接之后，即可进行软件编程实现，CAN 收发通讯的软件流程如下：



2.3.2 初始化系统时钟/GPIO

本示例中系统时钟采用 64M，PCLK 为 64M。CAN1-TX 为 PA12，CAN1-RX 为 PA11。

2.3.3 配置 CAN 工作模式/参数

因整车网络中 CAN 波特率通常为 250K 和 500K，故此例中以 500K 波特率为例进行配置。

参考 1.6 章节中波特率的计算方法，如设置波特率为 500K，系统时钟 64MHz, APB 时钟为 64MHz，则 BRP=7，TQ=2*(7+1)*(1/64M)= 16*(1/64M)= 1/4M，设置 TSEG1=2，TSEG2=3，则数据位时间=1+(2+1)+(3+1)=8TQ，则 baud=1/8TQ=1/(8*1/4M)= 1/(2M)=500Kbps。

注：配置波特率等参数只能在复位模式下配置，配置完成后将工作模式设置为正常模式。

2.3.3 配置 CAN 过滤方式

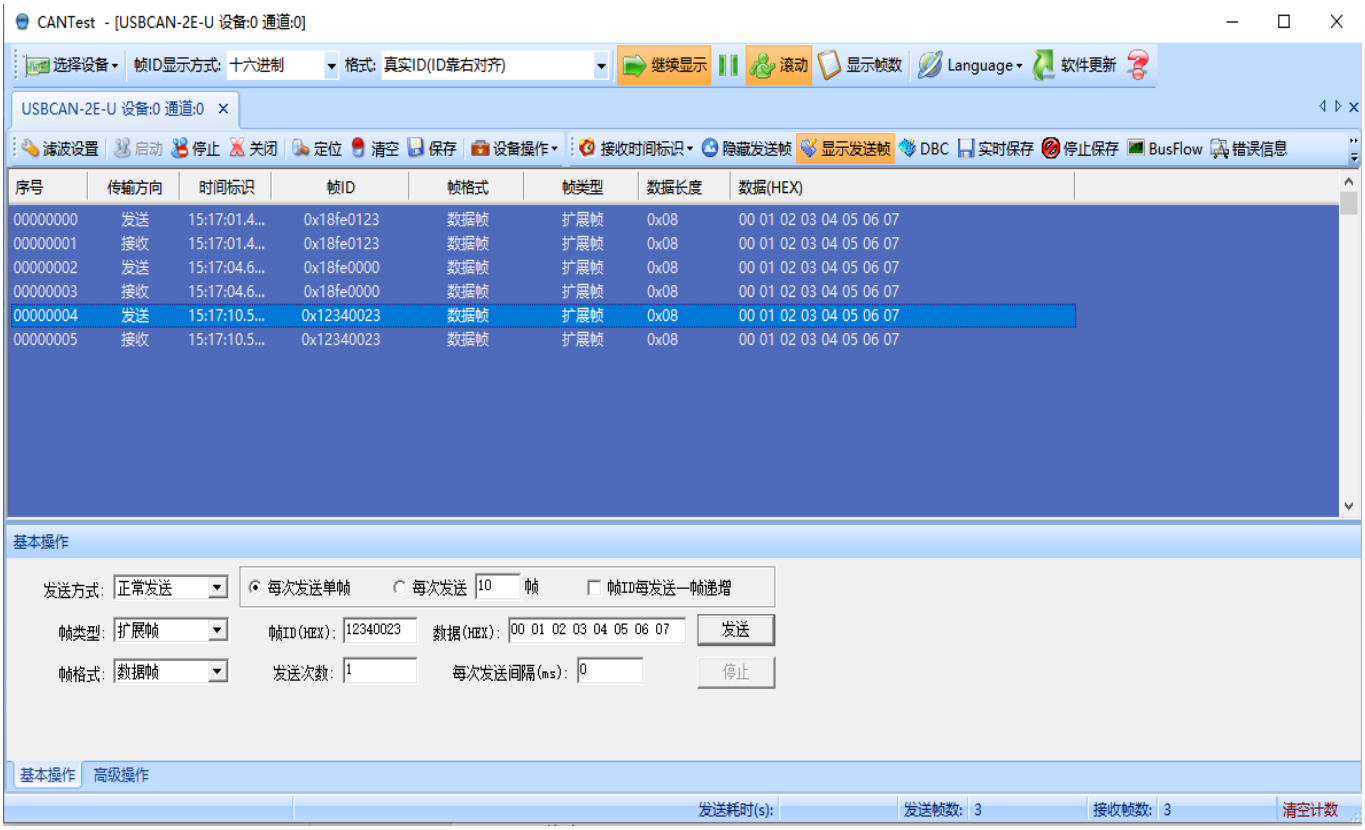
在 CAN 协议中，消息的标识符与节点地址无关，但与消息内容有关。因此，发送节点将报文广播给所有接收器时，接收节点会根据报文标识符的值来确定软件是否需要该消息，为了简化软件的工作，ACM32xxx 的 CAN 外设接收报文前会先使用过滤器检查，只接收需要的报文到 FIFO 中。

过滤方式见 1.5 章节“接收过滤”，本示例中采用双过滤模式，可以过滤扩展帧 29 位 ID 中的 13 到 28 位，通过设置 ACR 和 AMR 寄存器可以接收固定 ID 格式的报文。

本示例中参考商用车协议 J1939，报文 ID 格式多为 0x18FExxxx 的格式，将过滤报文 ID 设置为 0x18fexxxx 和 0x1234xxxx，即 CAN 节点只会接收这两种 ID 格式的扩展帧报文，其他 ID 的扩展帧报文将被自动过滤。

2.3.3 发送接收报文

完成上述过程，即可实现 CAN 报文收发，数据发送和接收请参考 1.2 和 1.3 章节，本示例中 ACM32xxx 的 CAN 节点会自动发送接收到的 CAN 报文，因此利用周立功 cantest 软件即可监控报文收发过程，如图所示。



三、CAN 模块芯片版本差异

3.1 F403/FP401/A403

本系列芯片分为两个版本，EFALSH NVR 区域地址 0x00080268 的 32bit 数据，表示芯片版本号。

- 0xFFCF0030 表示 A 版本，
- 0xFECF0130 表示 B 版本。

  对于 B 版本芯片 CAN1 模块，OCR 寄存器的 bit4 表示 CAN1 模块内部的 TX clock 的状态，当 TX clock=1 时，才能进行 CAN1 消息帧的发送。

  对于 B 版本芯片 CAN2 模块，CDR 寄存器的 bit2 表示 CAN2 模块内部的 TX clock 的状态，当 TX clock=1 时，才能进行 CAN2 消息帧的发送。

3.2 F0X0/A070/WB15

本系列芯片分为两个版本，EFALSH NVR 区域地址 0x00080268 的 32bit 数据，表示芯片版本号。

- 0xFFDE0021 表示 A 版本，
- 0xFEDE0121 表示 B 版本。

对于 B 版本芯片 CAN 模块, OCR 寄存器的 bit4 表示 CAN 模块内部的 TX clock 的状态，当 TX clock=1 时，才能进行 CAN 消息帧的发送。

联系我们

公司：上海爱信诺航芯电子科技有限公司

地址：上海市闵行区合川路 2570 号科技绿洲三期 2 号楼 702 室

邮编：200241

电话：+86-21-6125 9080

传真：+86-21-6125 9080-830

Email： Service@AisinoChip.com

Website： www.aisinochip.com

版本维护

版本	日期	作者	描述
V1.0	2021-03-05	Aisinochip	初始版
V1.1	2022-02-09	Aisinochip	增加Bus off恢复章节描述
V1.2	2023-02-09	Aisinochip	添加对A070系列芯片支持