SPC1169 CAN 使用指南

版本 A/0 – 2023 年 4 月

概述

控制器局域网（CAN）为串行通讯协议，不仅通信速度快，而且还能有效地支持具有很高安全等级的分布实时控制。在汽车电子行业里，使用 CAN 连接发动机控制单元、传感器、防刹车系统、也可以将 CAN 安装在卡车本体的电子控制系统里，诸如车灯组、电气车窗等等，用以代替接线配线装置，Spintrol 的 SPC1169 芯片就内建了一个 CAN 单元。

目录

图片列表

表格列表

版本历史

版本	日期	作者	状态	变更
A/0	2023 年 4 月 17 日	XuQing He	Released	首次发布。

术语或缩写

术语或缩写	描述
MCU	Microcontroller Unit，微控制器单元

1 CAN 特性

SPC1169 内建一个 CAN 单元，且具有以下特点：

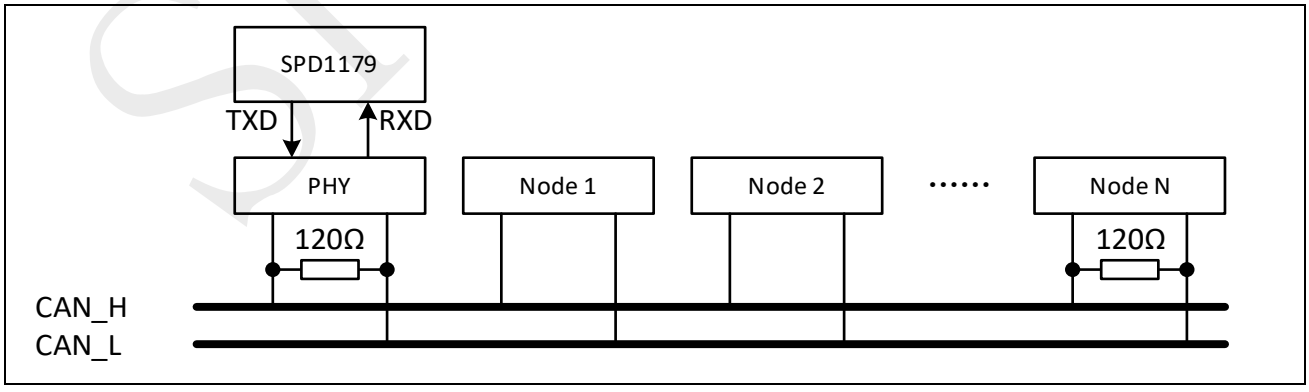
- 具有实时性强、传输距离较远、抗电磁干扰能力强、成本低等优点；
- 采用双线串行通信方式，检错能力强，可在高噪声干扰环境中工作；
- 具有优先权和仲裁功能，多个控制模块通过 CAN 控制器挂到 CAN-bus 上，形成多主机局部网络；
- 具有 64 个 Mailbox，每个有独立的 ID 过滤规则，可根据报文的 ID 决定接收或屏蔽该报文；
- 可靠的错误处理和检错机制；
- 发送的信息遭到破坏后，可自动重发；
- 节点在错误严重的情况下具有自动退出总线的功能；
- 报文不包含源地址或目标地址，仅用标志符来指示功能信息、优先级信息；
- 支持 CAN 总线错误的记录；
- 支持 32 位时间戳；

SPC1169 的 CAN 单元可以用作以下功能：

- 汽车发动机控制单元、传感器、防刹车系统；
- 汽车车灯组、电气车窗等控制系统；

CAN 模块与其他 CAN 设备连接示意图如图 1-1 所示。使用 SPC1169 时需要外部接个 PHY 芯片和 120Ω 的终端电阻连接到 CAN_H 和 CAN_L 一对双绞线上。CAN 总线的电平状态只有“显性”和“隐性”，“显性”代表低电平，“隐性”代表高电平，高低电平就是由 CAN_H 和 CAN_L 这对差分信号决定。

图 1-1：CAN 模块连接示意图



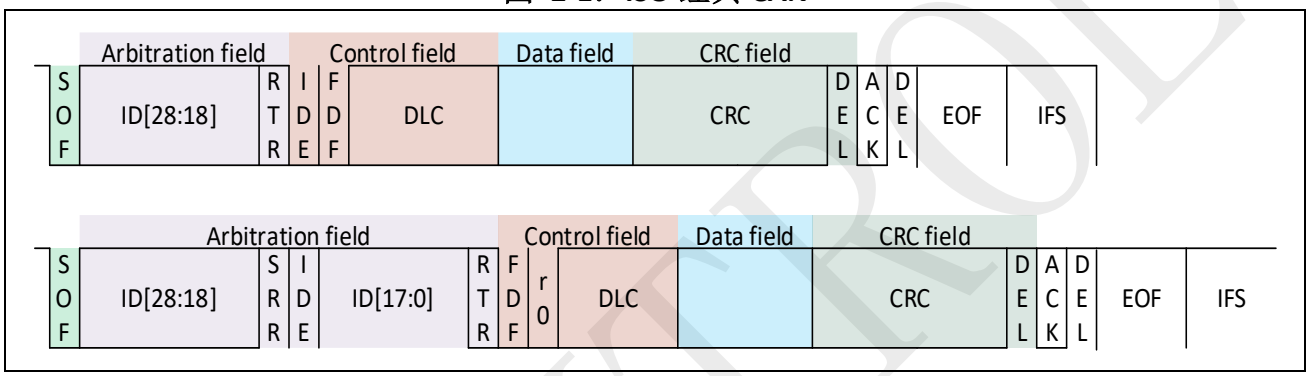
2 CAN 帧格式

SPC1169 支持 ISO（国际标准化）CAN 协议和 NONISO（非国际标准化）CAN 协议两种协议，ISO CAN 协议和 NONISO CAN 协议的切换可以通过 CANCTL 寄存器的 NONISO 位段进行配置。

2.1 ISO 经典 CAN

ISO 经典 CAN 支持标准帧和扩展帧两种帧格式，其标识符分别是 11 位和 29 位。

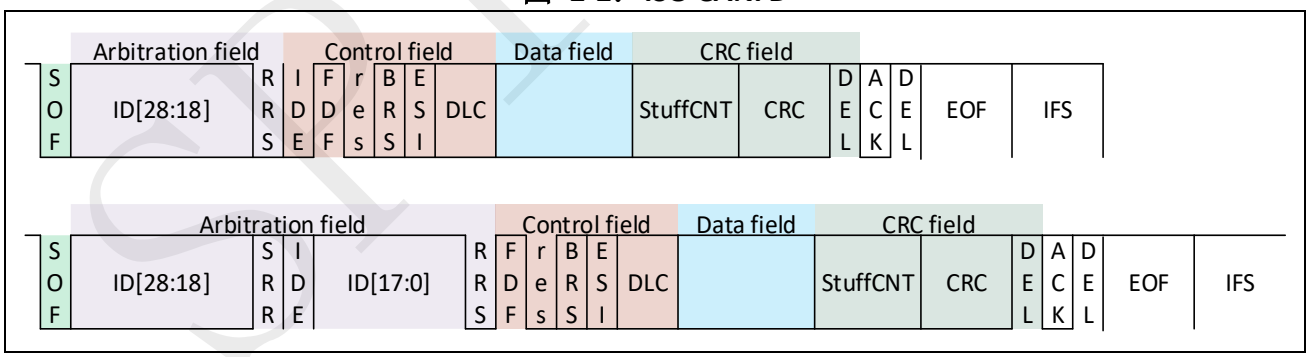
图 2-1：ISO 经典 CAN



2.2 ISO CANFD

ISO CANFD 支持标准帧和扩展帧两种帧格式，且仅支持数据帧的传输。

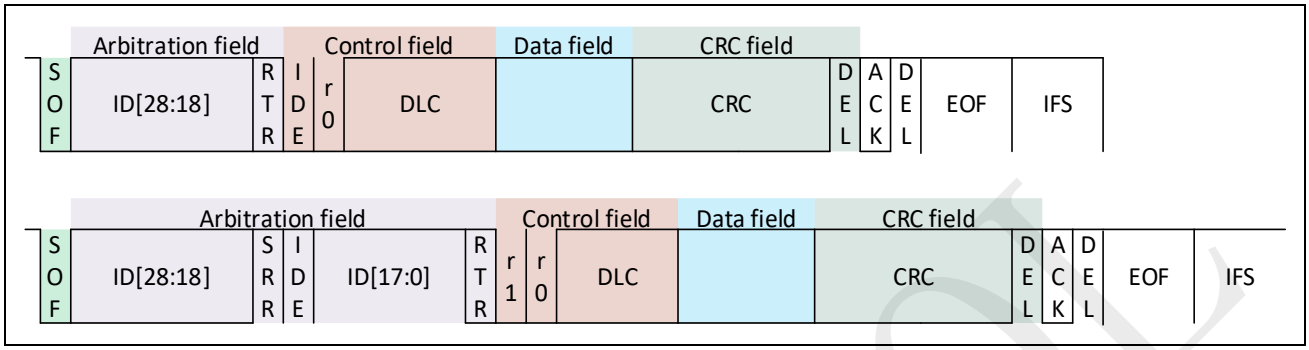
图 2-2：ISO CANFD



2.3 NONISO 经典 CAN

NONISO 经典 CAN 支持标准帧和扩展帧两种帧格式，与 ISO 的经典 CAN 帧格式相同。

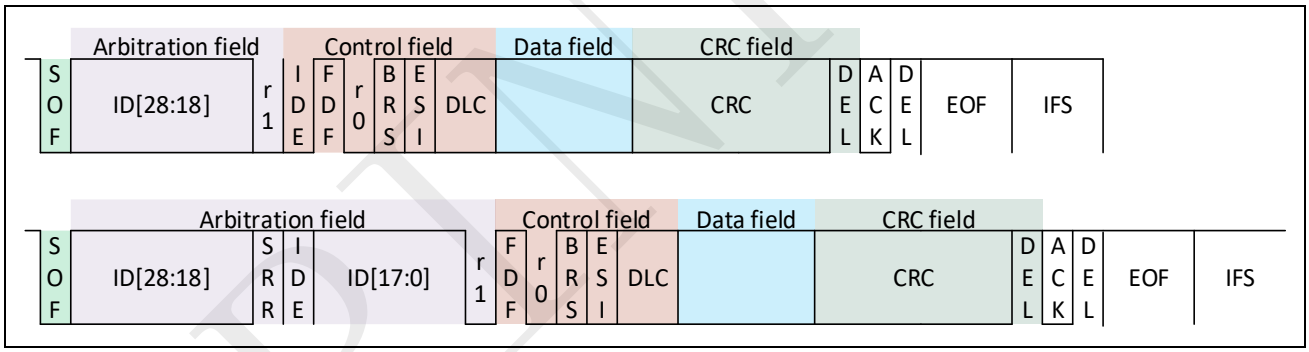
图 2-3：NONISO 经典 CAN



2.4 NONISO CANFD

NONISO CANFD 支持标准帧和扩展帧两种帧格式，且仅支持数据帧的传输。与 ISO CANFD 相比，NONISO CANFD 的 CRC 位段没有填充位段（stuff count），且采用了不同的 CRC 算法。

图 2-4：NONISO CANFD



3 协议帧解析

CAN 协议和 CANFD 协议的物理层结构相同，只更新了协议，CAN 的最大速率为 1Mbps，CANFD 的快速阶段的速率可达 5Mbps，慢速阶段的速率最高为 1Mbps。

CAN 支持标准帧和扩展帧两种帧格式，分别是标准帧格式和扩展帧格式，这两种帧格式的最大区别是标准帧格式具有 11 位识别符，而扩展帧格式有 29 位识别符。以下针对 ISO 的 CAN 的协议帧进行详细介绍。

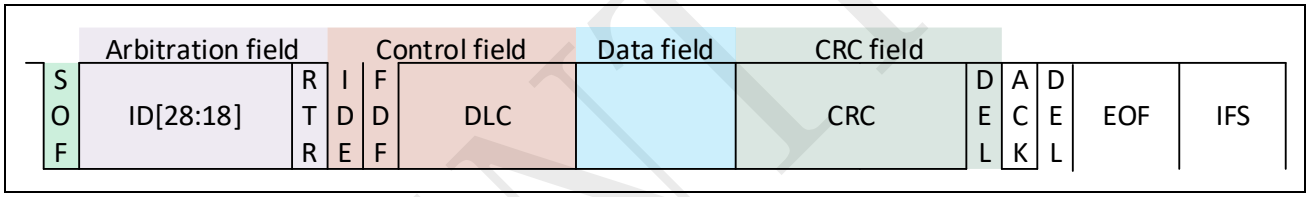
3.1 ISO 经典 CAN 协议帧

3.1.1 标准帧格式

3.1.1.1 数据帧

标准数据帧由 7 个不同的域组成：帧起始、仲裁域、控制域、数据域、CRC 域、应答域、帧结尾。

图 3-1：标准格式数据帧



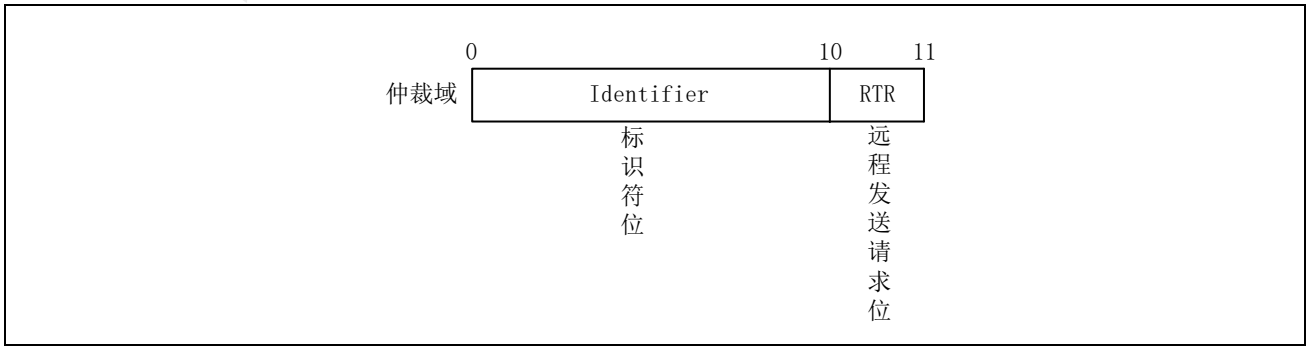
帧起始（Start Of Frame-SOF）： 标志数据帧和远程帧的起始。

仲裁域（Arbitration Field）： 包括标识符位和远程发送请求位。

标识符位：CAN 接收器通过标识符来过滤数据帧，ID28 为最高权重位，ID18 为最低权重位，按照 ID28~ID18 的顺序进行传输。CAN 协议规定，前 7 位最高权重位（ID28~ID22）不能都为“隐性”信号。

远程发送请求位：用于区分是数据帧还是远程帧，“显性”信号代表数据帧，“隐性”信号代表远程帧。

图 3-2：标准格式数据帧-仲裁域



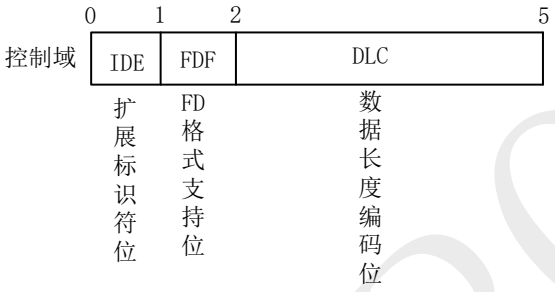
控制域（Control Field）：包含扩展标识符位、保留位和数据长度编码位。

扩展标识位：用来表示该帧是标准格式还是扩展格式，标准格式 IDE 位为“显性”。

FD 格式支持位：用来区分是经典 CAN 格式的帧还是 CANFD 的帧，为“显性”信号代表经典 CAN 的帧，为“隐性”信号代表 CANFD 的帧。

数据长度编码位：实际发送数据长度（以字节为单位）。

图 3-3：标准格式数据帧-控制域



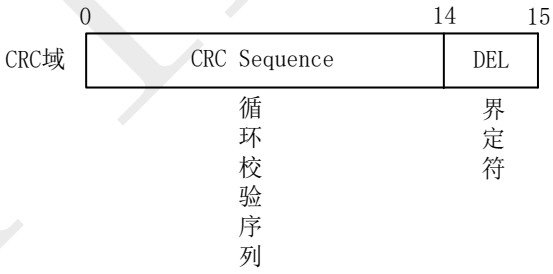
数据域（Data Field）：发送数据的内容，最多发送 8 个字节。

CRC 域（CRC Field）：包含循环校验序列和界定符。

循环校验序列：用于校验传输是否正确。

界定符：“隐性”信号表示循环校验序列的结束。

图 3-4：标准格式数据帧-CRC 域

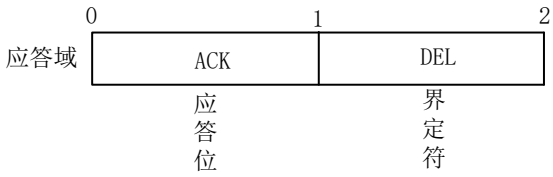


应答域（ACK Field）：包含应答位和界定符。

应答位： ACK 位被置位，表示收到正确的 CRC 校验序列。

界定符：“隐性”信号，应答位后需要一个“隐性”信号。

图 3-5：标准格式数据帧-应答域



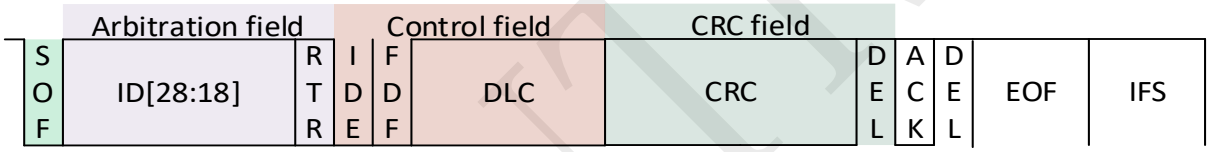
帧结尾（End of Frame-EOF））：全是隐性信号，表示帧的结束。

3.1.1.2 远程帧

标准远程帧由 6 个不同的域组成：帧起始、仲裁域、控制域、CRC 域、应答域、帧末尾。

远程帧和数据帧不同的是远程帧没有数据域，以及仲裁域的远程发送请求位（RTR）信号不同，远程帧 RTR 位是“隐性”，数据帧 RTR 位是“显性”。

图 3-6：标准格式远程帧

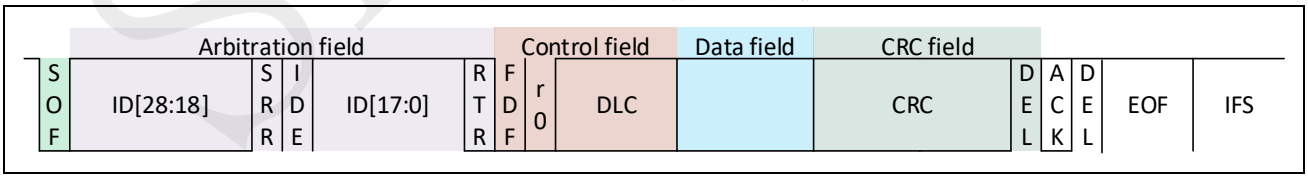


3.1.2 扩展格式

3.1.2.1 数据帧

扩展数据帧由 7 个不同的域组成：帧起始、仲裁域、控制域、数据域、CRC 域、应答域、帧结尾。

图 3-7：扩展格式远程帧



帧起始（Start Of Frame-SOF）：标志数据帧和远程帧的起始。

仲裁域（Arbitration Field）：包括基本标识符位、替代远程请求位（Substitute Remote Request BIT）、识别符扩展位、扩展标识符位和远程发送请求位。

基本标识符位：CAN 接收器通过标识符来过滤数据帧，按 ID28 到 ID18 的顺序发送。

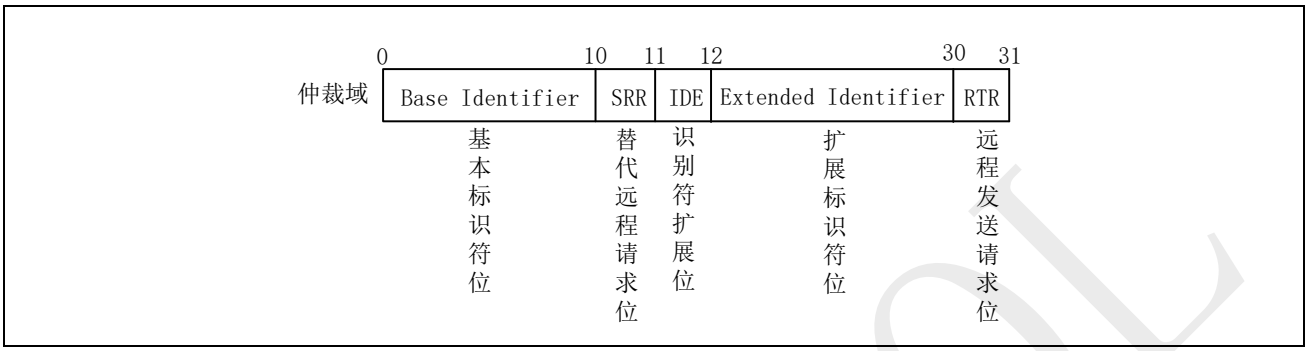
替代远程请求位：SRR 是一个“隐性”位。它代替标准帧 RTR 位置。

识别符扩展位：扩展格式 IDE 位为“隐性”。

扩展标识符位：CAN 接收器通过标识符来过滤数据帧，它按 ID17 到 ID0 顺序发送。

远程发送请求位：用于区分是数据帧还是远程帧，“显性”信号代表数据帧，“隐性”信号代表远程帧。

图 3-8：扩展格式数据帧-仲裁域



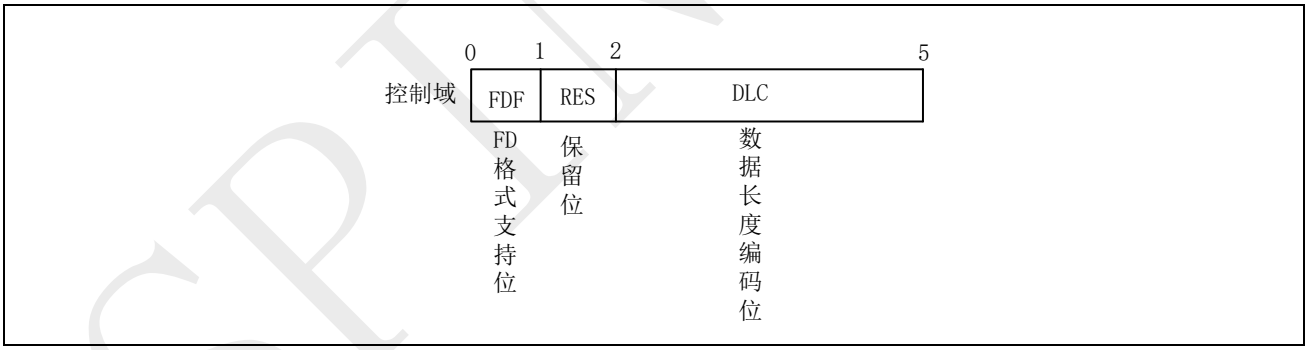
控制域（Control Field）：包含 FD 格式支持位、保留位和数据长度编码位。

FD 格式支持位：用来区分是经典 CAN 格式的帧还是 CANFD 的帧，为“显性”信号代表经典 CAN 的帧，为“隐性”信号代表 CANFD 的帧。

保留位：保留，以后使用。

数据长度编码位：实际发送数据长度（以字节为单位）。

图 3-9：扩展格式数据帧-控制域



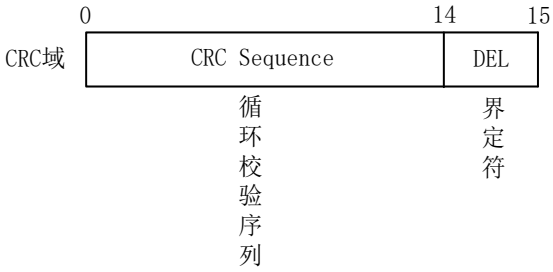
数据域（Data Field）：发送数据的内容，最多发送 8 个字节

CRC 域（CRC Field）：包含循环校验序列和界定符。

循环校验序列：用于校验传输是否正确。

界定符：“隐性”信号表示循环校验序列的结束。

图 3-10：扩展格式数据帧-CRC 域

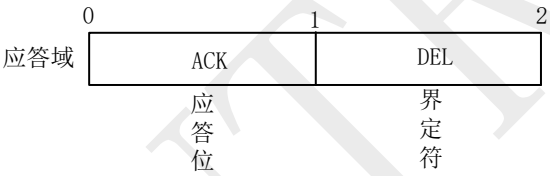


应答域（ACK Field）：包含应答位和界定符。

应答位：ACK 位被置位，表示收到正确的 CRC 校验序列。

界定符：“隐性”信号，应答位后需要一个隐性信号。

图 3-11：扩展格式数据帧-应答域



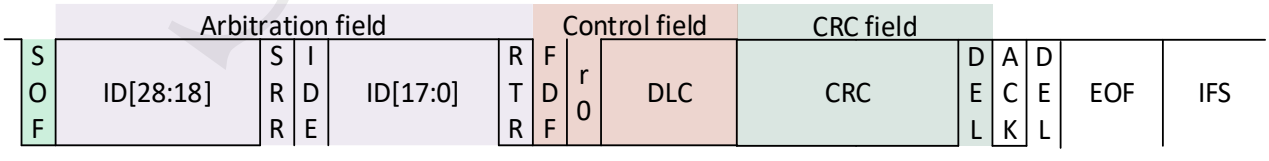
帧结尾（End of Frame-EOF））：全是隐性信号，表示帧的结束。

3.1.2.2 远程帧

扩展远程帧由 6 个不同的域组成：帧起始、仲裁域、控制域、CRC 域、应答域、帧末尾。

远程帧和数据帧不同的是远程帧没有数据域，以及仲裁域的远程发送请求位（RTR）信号不同，远程帧 RTR 位是“隐性”，数据帧 RTR 位是“显性”。

图 3-12：CAN 协议扩展远程帧帧

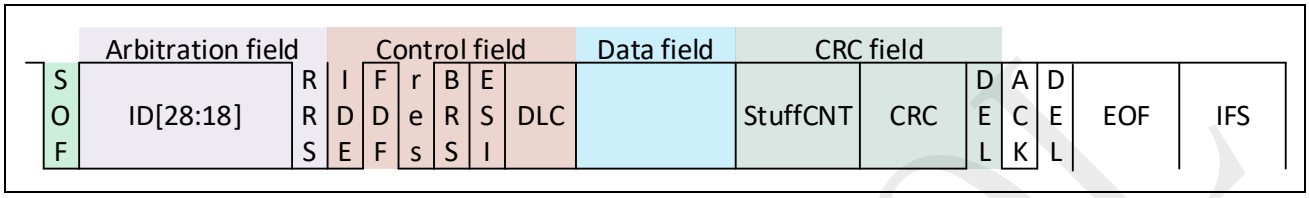


3.2 CANFD 协议帧格式

3.2.1 标准格式

标准数据帧由 7 个不同的域组成：帧起始、仲裁域、控制域、数据域、CRC 域、应答域、帧结尾。

图 3-13：CANFD 协议标准数据帧



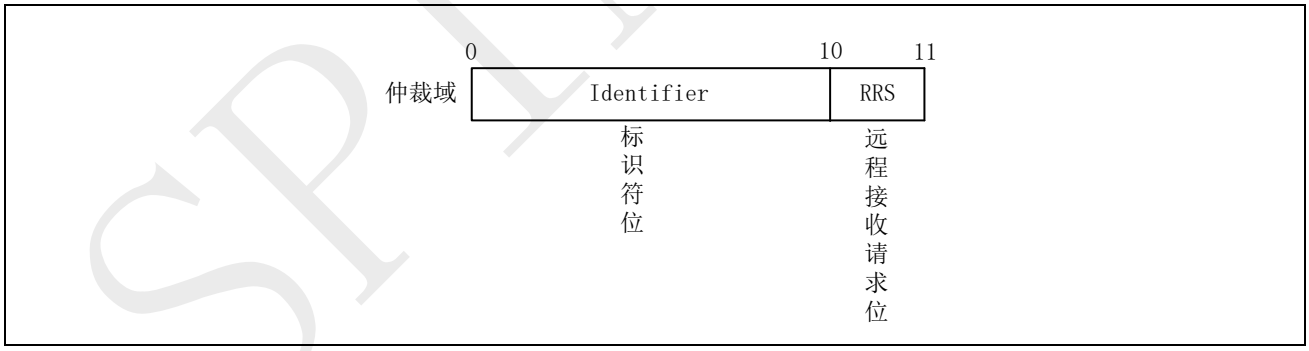
帧起始（Start Of Frame-SOF）：标志数据帧和远程帧的起始。

仲裁域（Arbitration Field）：包括标识符位和替代远程请求位（Substitute Remote Request BIT）。

标识符位： CAN 接收器通过标识符来过滤数据帧，ID28 位最高权重位，ID18 位最低权重位，按照 ID28~ID18 的顺序进行传输。CAN 协议规定，前 7 位最高权重位（ID28~ID22）不能都为“隐性”信号。

远程接收请求位：用于远程帧请求，“显性”信号表示请求接收者发送一个远程帧；“隐性”信号表示请求发送者发送一个远程帧。

图 3-14：标准格式数据帧-仲裁域



控制域（Control Field）：包含扩展标识符位、FD 格式指示位、保留位、位速率转换开关位、错误状态指示位和数据长度编码位。

扩展标识位：用来表示该帧是标准格式还是扩展格式，标准格式 IDE 位为“显性”信号，扩展格式为“隐性”信号。

FD 格式指示位：用来区分是经典 CAN 格式的帧还是 CANFD 的帧，为“显性”信号代表经典 CAN 的帧，为“隐性”信号代表 CANFD 的帧。

保留位：保留，以后使用。

位速率转换开关位：当 BRS 为“隐性”信号时，表示在 BRS 位的采样点，将会切换到高速传输，然后在 CRC 域界定符的采样点，再切换回低速传输。

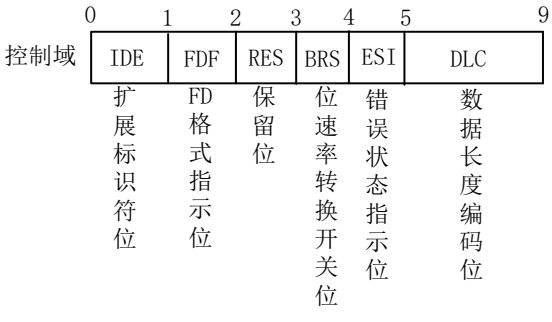
错误状态指示位：ESI 是指示发送节点的错误状态的标志，当发送节点的错误状态是激活时，发送“隐性”信号，如果错误状态未激活时，发送“显性”信号。通过 ESI 位的状态，所有节点都可以确认当前的传输节点的错误状态。而在 CAN 帧中，无法得知其传输节点的错误状态。

数据长度编码位：实际发送数据长度，最多可以发送 64 个字节，其配置如表 3-1 所示。

表 3-1：CANFD 协议数据长度编码规则

数据长度	DLC 值
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
12	9
16	10
20	11
24	12
32	13
48	14
64	15

图 3-15：标准格式数据帧-控制域



数据域（Data Field）：发送数据的内容，最多发送 64 个字节。

CRC 域（CRC Field）：包含 Stuff Count 位、循环校验序列和界定符。

Stuff Count 位：Stuff Count 由格雷码计算计算结果和奇偶校验结果组成。

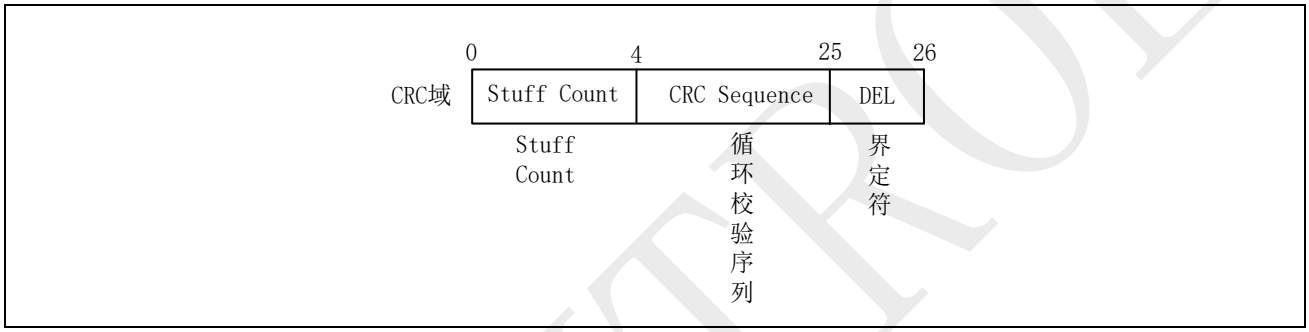
格雷码计算：CRC 区域之前的填充位数除以 8，得到的余数（Stuff bit count modulo 8）进行格雷码计算得到的值（Bit0-2）。

奇偶校验：通过格雷码计算后的值的奇偶校验（Bit3）。

循环校验序列：用于校验传输是否正确。

界定符：隐性信号表示循环校验序列的结束。

图 3-16：标准格式数据帧-CRC 域

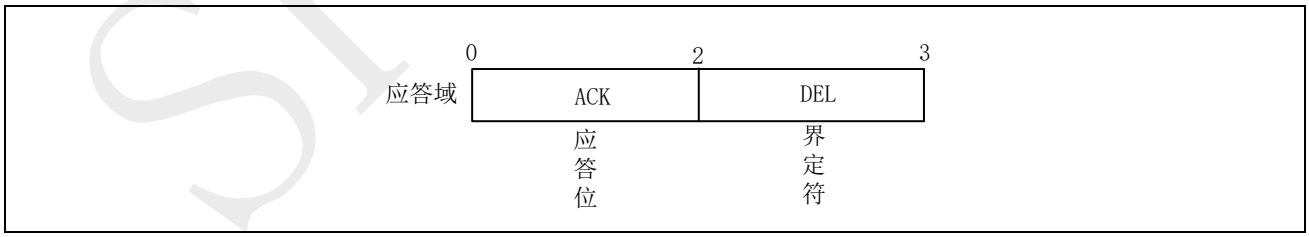


应答域（ACK Field）：包含应答位和界定符。

应答位：ACK 位为“显性”信号，表示收到正确的 CRC 校验序列，由从高速阶段到慢速阶段时，时钟切换会引起收发器相移和总线传播延迟。为了补偿其相移和延迟，相比传统的 CAN，在 CANFD 中多加了这额外的 1 位时间。

界定符：“隐性”信号，应答位后需要一个“隐性”信号。

图 3-17：标准格式数据帧-应答域

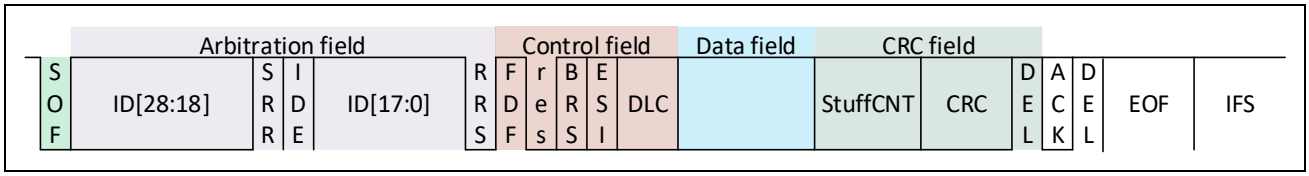


帧结尾（End of Frame-EOF））：全是隐性信号，表示帧的结束。

3.2.2 扩展格式

标准数据帧由 7 个不同的域组成：帧起始、仲裁域、控制域、数据域、CRC 域、应答域、帧结尾。

图 3-18：CANFD 协议标准数据帧



帧起始（Start Of Frame-SOF）：标志数据帧和远程帧的起始。

仲裁域（Arbitration Field）：包括基本标识符位、替代远程请求位（Substitute Remote Request BIT）、识别符扩展位、扩展标识符位和远程发送请求位。

基本标识符位：CAN 接收器通过标识符来过滤数据帧，按 ID28 到 ID18 的顺序发送。

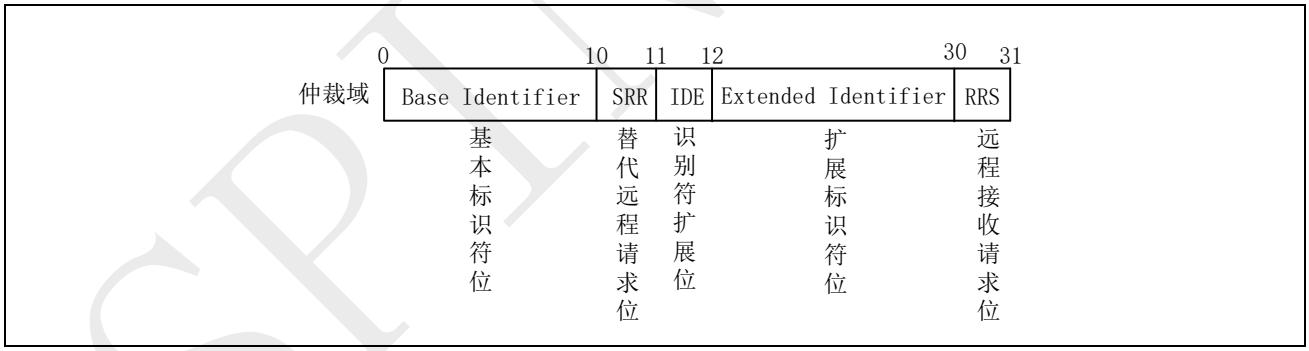
替代远程请求位：SRR 是一个“隐性”信号。它代替标准帧 RRS 位置。

识别符扩展位：扩展格式 IDE 位为“隐性”信号

扩展标识符位：CAN 接收器通过标识符来过滤数据帧，它按 ID17 到 ID0 顺序发送。

远程接收请求位：用于远程帧请求，“显性”信号表示请求接收者发送一个远程帧；“隐性”信号表示请求发送者发送一个远程帧。

图 3-19：标准格式数据帧-仲裁域



控制域（Control Field）：包含 FD 格式指示位、保留位、位速率转换开关位、错误状态指示位和数据长度编码位。

FD 格式指示位：用来区分是经典 CAN 格式的帧还是 CANFD 的帧，为“显性”信号代表经典 CAN 的帧，为“隐性”信号代表 CANFD 的帧。

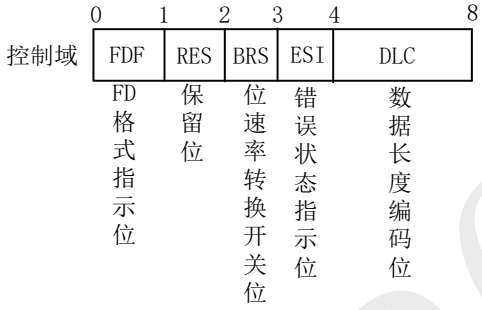
保留位：保留，以后使用。

位速率转换开关位：当 BRS 为“隐性”信号时，表示在 BRS 位的采样点，将会切换到高速传输，然后在 CRC 域界定符的采样点，再切换回低速传输。

错误状态指示位：ESI 是指示发送节点的错误状态的标志，当发送节点的错误状态是激活时，发送“隐性”信号，如果错误状态未激活时，发送“显性”信号。通过 ESI 位的状态，所有节点都可以确认当前的传输节点的错误状态。而在 CAN 帧中，无法得知其传输节点的错误状态。

数据长度编码位：实际发送数据长度，最多可以发送 64 个字节，其配置如表 3-1 所示。

图 3-20：标准格式数据帧-控制域



数据域（Data Field）：发送数据的内容，最多发送 64 个字节。

CRC 域（CRC Field）：包含 Stuff Count 位、循环校验序列和界定符。

Stuff Count 位：Stuff Count 由格雷码计算计算结果和奇偶校验结果组成。

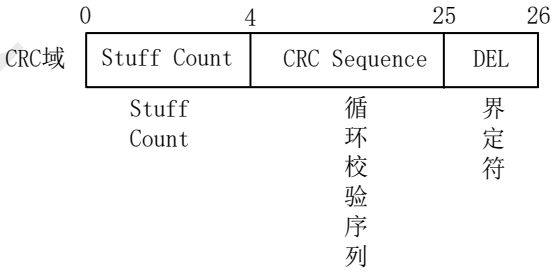
格雷码计算：CRC 区域之前的填充位数除以 8，得到的余数（Stuff bit count modulo 8）进行格雷码计算得到的值（Bit0-2）。

奇偶校验：通过格雷码计算后的值的奇偶校验（Bit3）。

循环校验序列：用于校验传输是否正确。

界定符：“隐性”信号表示循环校验序列的结束。

图 3-21：标准格式数据帧-CRC 域

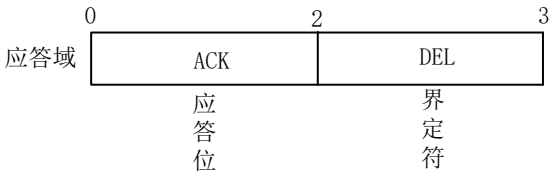


应答域（ACK Field）：包含应答位和界定符。

应答位：ACK 位为“显性”信号，表示收到正确的 CRC 校验序列，由从高速阶段到慢速阶段时，时钟切换会引起收发器相移和总线传播延迟。为了补偿其相移和延迟，相比传统的 CAN，在 CANFD 中多加了这额外的 1 位时间。

界定符：“隐性”信号，应答位后需要一个“隐性”信号。

图 3-22：标准格式数据帧-应答域

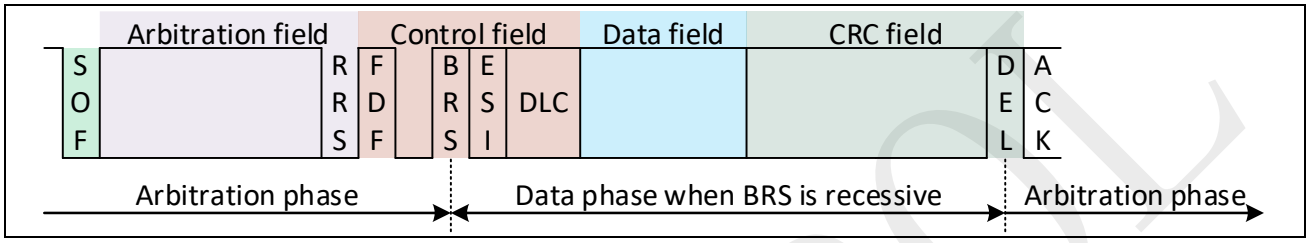


帧结尾（End of Frame-EOF））： 全是“隐性”信号，表示帧的结束。

4 慢速阶段和快速阶段

对于启用了速率切换的 ISO CANFD 或 NONISO CANFD 帧传输，都具有两个不同速率的阶段。如图 4-1 所示，在控制域的 BRS 位采样点之后到 CRC 域的 DEL 位采样点之前的传输称为快速阶段，所有其他位传输称为慢速阶段，并且快速阶段的速率通常比慢速阶段的速率快。对于禁用了速率切换的 CANFD 帧或经典 CAN 帧，快速阶段的速率和慢速阶段的速率一样。

图 4-1：慢速阶段和快速阶段



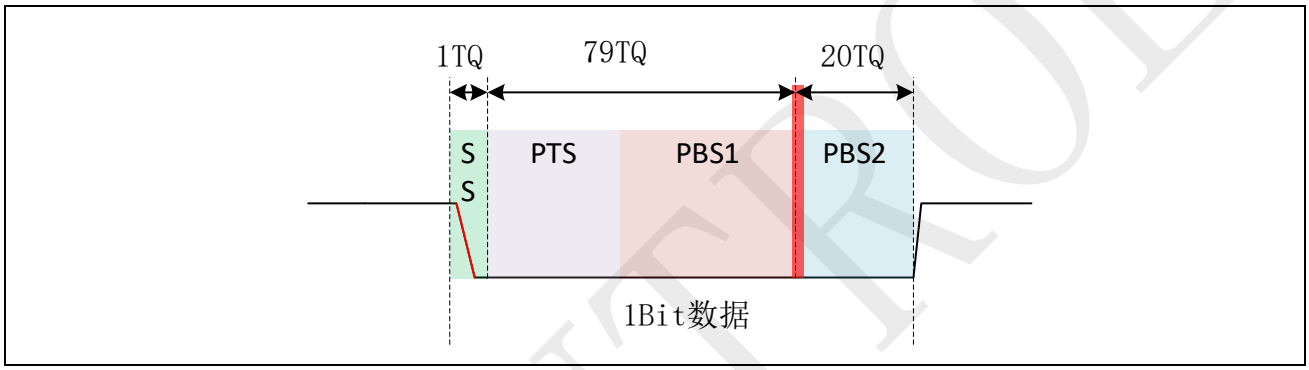
5 位同步基本概念

由于 CAN 通信没有时钟信号，属于异步通信，所以连接到总线上的各个节点以约定好的波特率进行传输，但 CAN 通信还会使用硬同步和再同步的方式来进行抗干扰、吸收误差、实现对总线电平信号的正确采样，确保通信的正常。

如图 5-1 所示，一个比特的数据可以分为 4 段：同步段（SS）、传播时间段（PTS）、相位缓冲段 1（PBS1）、相位缓冲段 2（PBS2）。

这些段是称为 Time Quantum（TQ）的最小时间单位构成，PTS 和 PBS1 段的时间由 CANNBTQ 寄存器的 NSEG1 位段控制，PBS2 段的时间由 CANNBTQ 寄存器的 NSEG2 位段控制。

图 5-1：位时序

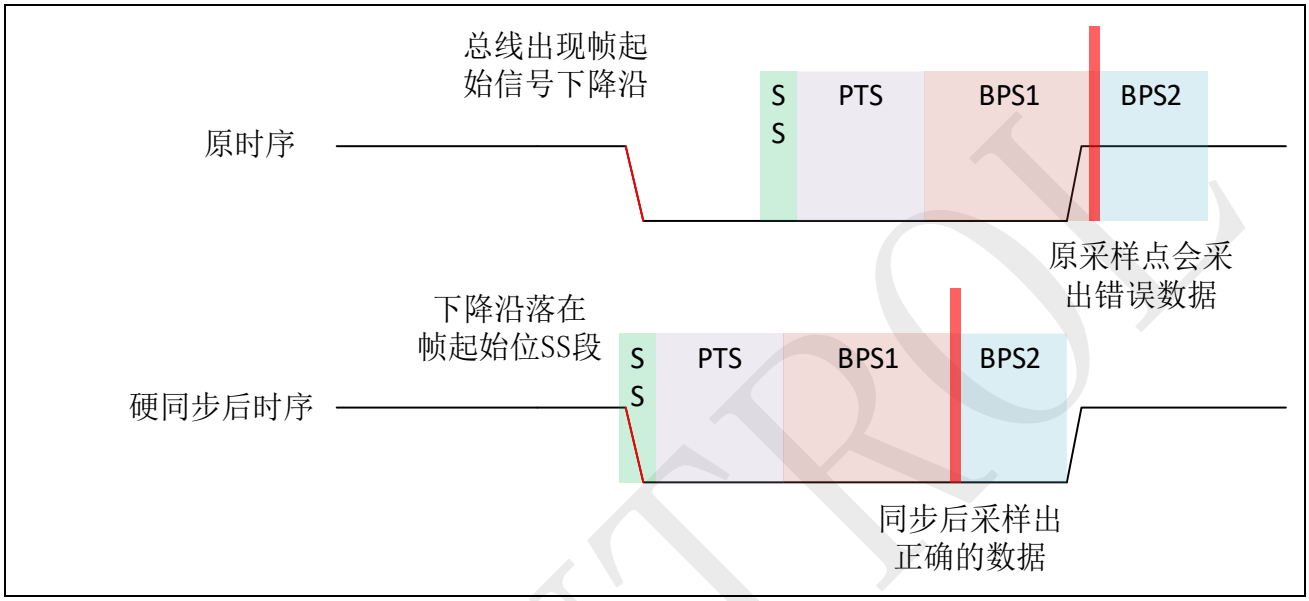


在 CAN 通信中有两种位同步机制，分别是硬同步和再同步。

5.1 硬同步

硬同步在总线空闲状态检测出第一个下降沿时（对应报文的 SOF 下降沿）进行的同步调整。在检测到 SOF 的下降沿时，直接将此下降沿的位置调整到 SS 段，然后按照位时序对信号进行采样，达到同步的效果，硬同步的过程如图 5-2：硬同步所示。

图 5-2：硬同步



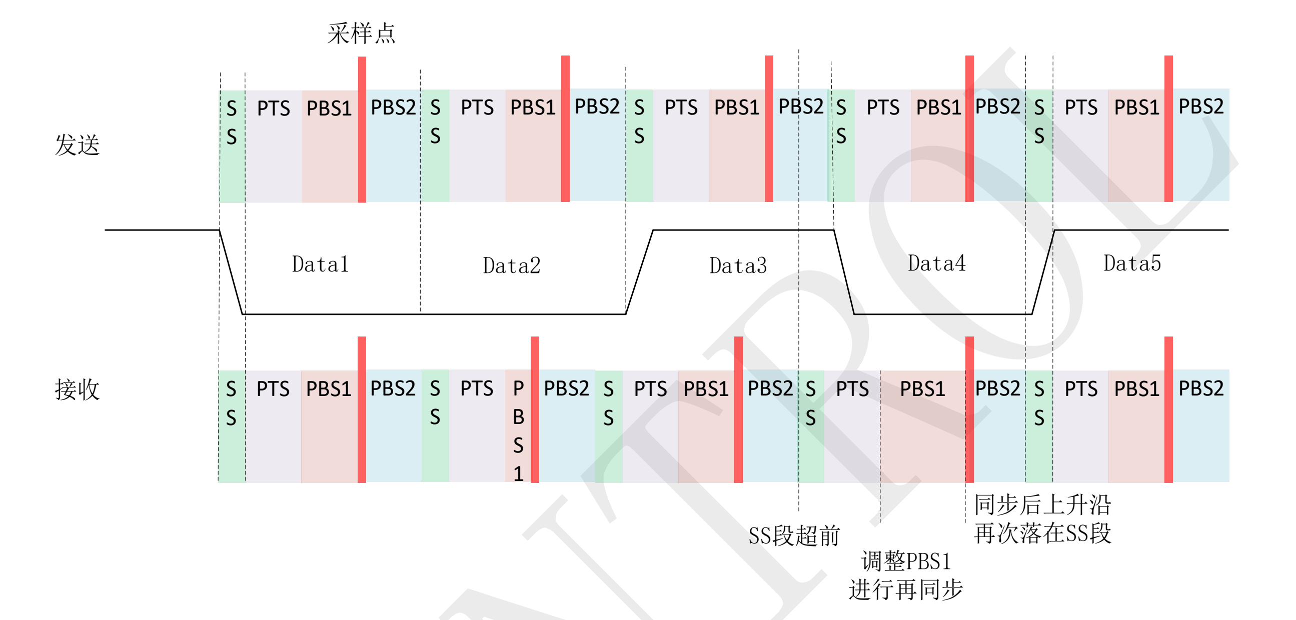
5.2 再同步

总线在空闲时检测出下降沿（帧起始）时进行硬同步后，但后续 CAN 报文数据位的同步则需要通过调整 BPS1 或者 BPS2 段时间来进行再同步。在接收过程中检测出总线上的下降沿来临时，通过加长 PBS1 段时间或缩短 PBS2 段时间进行同步。但如果发生了超出 SJW 值的误差时，最大调整量不能超过 SJW 值。

5.2.1 SS 段超前，延长 PBS1 进行同步

如图 5-3：再同步-SS 段超前所示，在传输过程中，接收节点在接收 Data2 数据时，由于传输过程中的干扰导致 Data2 的 PBS1 段的时长缩短影响后续数据的采样，当总线出现下降沿时，接收节点会调节 PBS1 段的时长进行恢复，PBS1 的时长只在同步的那一位有效，该位结束后，PBS1 会恢复预设值。

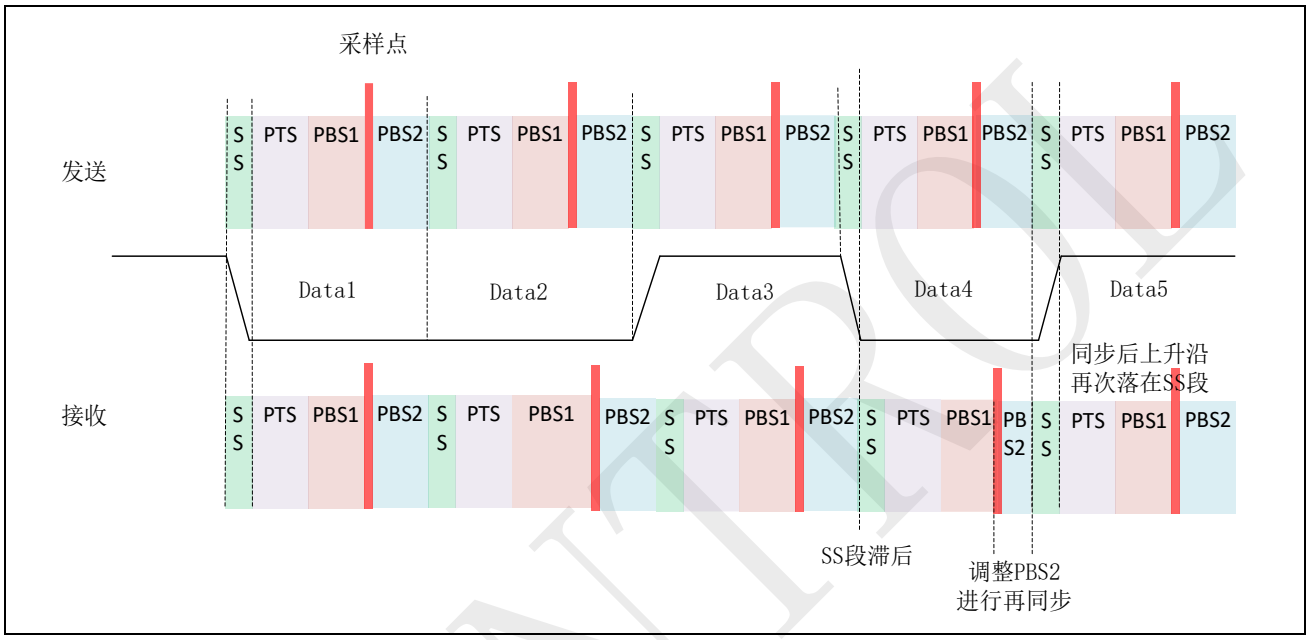
图 5-3：再同步-SS 段超前



5.2.2 SS 段滞后，缩短 PBS2 进行同步

如图 5-4：再同步-SS 段滞后所示，在传输过程中，接收节点在接收 Data2 数据时，由于传输过程中的干扰导致 Data2 的 PBS1 段的时长增大影响后续数据的采样，当总线出现下降沿时，接收节点会调节 PBS2 段的时长进行恢复，PBS2 的时长只在同步的那一位有效，该位结束后，PBS2 会恢复预设值。

图 5-4：再同步-SS 段滞后



6 CAN 的使用配置

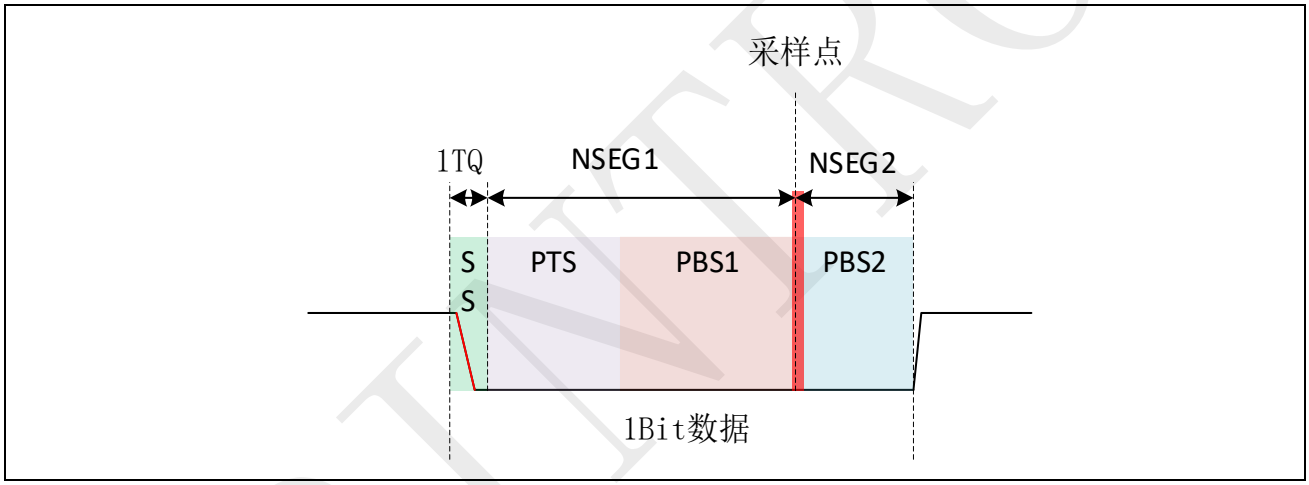
6.1 配置 CAN 使用的协议

典型 CAN 和 CANFD 只是对协议进行升级，物理层不变，主要区别是传输的速率不同、数据长度不同以及帧格式不同。CAN 的默认配置是使用典型的 CAN 协议，如果需要使用 CANFD1 协议需要对 CANCTL 寄存器的 FDEN 位进行配置。

6.2 配置 CAN 通信速率和采样点

典型 CAN 和 CANFD 的传输速率不一样，CAN 协议只有一种速率而 CANFD 协议具有两种速率，快速阶段和慢速阶段的速率不同。CAN 的最高可达 1Mbps，而 CANFD 的快速阶段的速率可达 5Mbps。

图 6-1：通信速率和采样点



如图 6-1 所示，一个 Bit 数据分为 4 个段，数据的采样在 PBS1 和 PBS2 段之间，其中 SS 段占用的时间位 1TQ，PTS 和 PBS1 段的时间可以通过 CANNBTQ 寄存器的 NSEG1 位段进行配置，可配置的范围为 2~256QT；PBS2 段的时间可以通过 CANNBTQ 寄存器的 NSEG2 位段进行配置，可配置的范围为 2~128QT；采样的分频系数可以通过 CANNBTQ 寄存器的 NBRP 位段进行配置，可配置的范围为 1~255；

通信速率f_1计算公式：

$$f_1 = \frac{f_m}{\left((NSEG1 + 1) + NSEG2\right) * NBRP}$$

采样点计算公式：

$$Sample\ Point = \frac{NSEG1 + 1}{NSEG1 + 1 + NSEG2}$$

f_m为 CAN 模块的时钟频率可以通过 CLOCK_GetModuleClock(CAN_MODULE)函数获取 CAN 模块的工作频率。

$NSEG1$为采样前的时间可以通过 CAN_SetNominalBitPhaseBufferSegment1()函数设置以及 CAN_GetNominalBitPhaseBufferSegment1()函数获取。

$NSEG2$为采样后的时间可以通过 CAN_SetNominalBitPhaseBufferSegment2()函数设置以及 CAN_GetNominalBitPhaseBufferSegment2()函数获取。

$NBRP$为波特率分频系数可以通过 CAN_SetNominalBitRatePrescaler()函数设置以及 CAN_GetNominalBitRatePrescaler()函数获取。

CANFD 协议的快速阶段的速率和慢速阶段的速率不同需要另外配置，但计算通信速率和采样点的方法与上面一致。采样前的时间可以通过 CANDBTQ 寄存器的 DSEG1 位段进行配置；采样后的时间可以通过 CANDBTQ 寄存器的 DSEG2 位段进行配置；波特率的分频系数可以通过 CANDBTQ 寄存器的 DBRP 位段进行配置；然后根据上述公式计算 CANFD 的速率和采样点。

6.3 配置 CAN 发送的信息

CAN 的信息配置都是通过调用 CAN_SetMessage(CAN_REGS *CANx, CAN_MessageTypeDef *pMsg)函数进行设置，其中 pMsg 为填充发送或者接收信息的变量，其 CAN_MessageTypeDef 定义如表 6-1。

表 6-1：CAN Message 配置

```
/**
 *  @brief  CAN mailbox type definition
 */
typedef struct
{
    /* Message definition */

    /*!< Frame format: STD,EXT,FD_STD,FD_EXT            */
    CAN_FrameFormatEnum     eFormat       ;

    /*!< Frame type  : data frame or remote frame        */
    CAN_FrameTypeEnum          eType          ;

    /*!< Message identifier                            */
    uint32_t                u32Id         ;

    /*!< Message identifier mask                        */
    uint32_t                u32IdMask      ;

    /*!< Enable mask RTR bit                           */
    FunctionalState           eMaskRtrEn     ;

    /*!< Enable mask IDE bit                           */
    FunctionalState           eMaskIdeEn     ;

    /*!< Message data length                           */
    uint8_t                 u8DataLen      ;

    /*!< Message data buffer pointer                    */
    uint8_t                 *pu8Data       ;
```

```
    /*!< Message data timestamp based on nominal bit time */
    uint32_t                u32Timestamp    ;

    /*!< Bit Rate Switch enable bit                    */
    FunctionalState          eBrs           ;

    /*!< Error State Indicator bit                     */
    FunctionalState          eEsi           ;

    /*!< ESI control bit when send mesage :
         0: send ESI with node error state;
         1: send ESI with eEsi value.
    */
    FunctionalState          eEsiCtlEn      ;

    /* Mailbox control definition */
    /*!< Mailbox ID                                 */
    uint8_t                 u8MBoxId       ;

    /*!< Transmission or reception message data          */
    CAN_DataDirectionEnum      eDataDir       ;

    /*!< Enable response remote frame                   */
    FunctionalState          eRmtRspEn      ;

    /*!< Enable transfer done interrupt                 */
    FunctionalState          eIntEn         ;

    /*!< Current mailbox is End Of Block, valid for reception data         */
    FunctionalState          eEobEn         ;

    /*!< Enable message overwrite, valid for reception data and EOB enabled */
    FunctionalState          eOverwriteEn    ;
} CAN_MessageTypeDef ;
```

eFormat：传输帧的格式，CAN_FORMAT_STD(CAN 协议的标准格式)、CAN_FORMAT_EXT(CAN 协议的扩展格式)、CAN_FORMAT_FD_STD(CANFD 协议的标准格式)、CAN_FORMAT_FD_EXT(CANFD 协议的扩展格式)

eType：传输帧的类型，CAN_FRAME_DATA（数据帧）、CAN_FRAME_REMOTE（远程帧）

u32Id：传输信息的标识符，用来过滤消息。

u32IdMask：传输信息的标识符屏蔽位。

eMaskRtrEn：远程帧传输请求位屏蔽位。

eMaskIdeEn：扩展标识符屏蔽位。

u8DataLen：发送消息的数据长度。

pu8Data：发送消息的数据地址。

u32Timestamp：消息数据的时间戳，只能读取。

eBrs：数据域速度切换使能位，对 CANFD 协议有效。

eEsi：错误状态指示器，指示当前帧是否处于错误状态，对 CANFD 协议有效。

eEsiCtlEn：错误状态指示器使能位，对 CANFD 协议有效。

u8MBoxId：使用邮箱的 ID 号。

eDataDir：消息传输的方向，CAN_MSG_DATA_RX(接收消息)、CAN_MSG_DATA_TX(发送消息)。

eRmtRspEn：使能远程帧自动响应，当在接收时使能，先发送远程帧，再等待接收数据帧；当在发送时使能，先等待接收远程帧，然后在邮箱中传输邮件。

eIntEn：使能传输完成中断。

eEobEn：当前邮箱为阻止结束，对接收数据有效。

eOverwriteEn：启用消息覆盖，对接收数据有效，启用 EOB。

在调用 CAN_SetMessage()函数配置完信息后，需要调用 CAN_EnableMailbox()函数使能邮箱进行发送或者接收数据，当接收或者发送数据成功后会对 CANMBOXMCTL 寄存器的 NEW 位进行置位，如果没有置位说明信息未传输成功。

7 CAN 实例

CAN 和 CANFD 只是对协议进行升级，物理层不变，主要区别是传输的速率不同、数据长度不同以及帧格式不同，以及 CANFD 不支持传输远程帧。如果使用 CAN 协议需要调用 CAN_DisableFDFormat()函数使能 CAN 协议，如果使用 CANFD 协议需要调用 CAN_EnableFDFormat()函数使能 CANFD 协议。

7.1 CAN 协议传输

7.1.1 发送标准帧和扩展帧数据

本示例中演示使用 CAN 协议依次发送标准格式远程帧、标准格式数据帧、扩展格式远程帧和扩展格式数据帧信息，传输的速率为 1Mbps，采样点为 80%，其配置流程如下：

- 调用 CAN_DisableFDFormat()函数失能 CANFD 协议，使用默认的典型 CAN 协议；
- 调用 CAN_DisableNonIsoMode()函数失能 NONISO 模式，使用默认的 ISO 模式；
- 调用 CAN_SetNominalBitRatePrescaler()、CAN_SetNominalBitPhaseBufferSegment1()、CAN_SetNominalBitPhaseBufferSegment2()函数，根据 6.2 章节计算 CAN 的通信速率和采样点；
- 配置发送标准格式的远程帧数据；

  eDataDir= CAN_MSG_DATA_TX 配置消息传输的方向为发送；

  u32Id = 0x11 配置传输消息的标识符为 0x11；

  u8MBoxId = 0 配置使用的邮箱的 ID 为 0；

  eFormat = CAN_FORMAT_STD 配置传输的消息为标准格式；

  eType = CAN_FRAME_REMOTE 配置传输的消息为远程帧类型；

  u8DataLen = 0 配置传输数据的长度为 0；

  pu8Data = NULL 配置传输的数据为空；
- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；
- 配置发送标准格式的数据帧数据；

  eDataDir= CAN_MSG_DATA_TX 配置消息传输的方向为发送；

  u32Id = 0x22 配置传输消息的标识符为 0x22；

  u8MBoxId = 1 配置使用的邮箱的 ID 为 1；

  eFormat = CAN_FORMAT_STD 配置传输的消息为标准格式；

  eType = CAN_FRAME_DATAE 配置传输的消息为数据帧类型；

  u8DataLen = CAN_DATA_LEN 配置传输数据的长度为 CAN_DATA_LEN；

  pu8Data = au8TxBuffer 配置传输的数据；
- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；
- 配置发送扩展格式的远程帧

eDataDir= CAN_MSG_DATA_TX 配置消息传输的方向为发送；

u32Id = 0x33 配置传输消息的标识符为 0x33；

u8MBoxId = 3 配置使用的邮箱的 ID 为 3；

eFormat = CAN_FORMAT_EXT 配置传输的消息为扩展格式；

eType = CAN_FRAME_REMOTE 配置传输的消息为远程帧类型；

u8DataLen = 0 配置传输数据的长度为 0；

pu8Data = NULL 配置传输的数据为空；

- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；
- 配置发送扩展格式的数据帧

eDataDir= CAN_MSG_DATA_TX 配置消息传输的方向为发送；

u32Id = 0x44 配置传输消息的标识符为 0x44；

u8MBoxId = 4 配置使用的邮箱的 ID 为 4；

eFormat = CAN_FORMAT_EXT 配置传输的消息为扩展格式；

eType = CAN_FRAME_DATAE 配置传输的消息为数据帧类型；

u8DataLen = CAN_DATA_LEN 配置传输数据的长度为 CAN_DATA_LEN；

pu8Data = au8TxBuffer 配置传输的数据；

- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；

发送标准帧和扩展帧数据

```
#include <stdio.h>
#include <stdlib.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif

#define DEBUG_INFO      1
#define CAN_DATA_LEN    8

uint8_t au8TxBuffer[64];

/* CAN Standard Remote frame message1 */
CAN_MessageTypeDef message1;
/* CAN Standard Data frame message2 */
CAN_MessageTypeDef message2;
/* CAN Extended Remote frame message3 */
CAN_MessageTypeDef message3;
/* CAN Extended Data frame message4 */
CAN_MessageTypeDef message4;

static void CAN_Clk_Enable(void)
{
    /* CAN Clk Enable */
    CLOCK_SetModuleDiv(CAN_MODULE, 1);
```

```
    CLOCK_EnableModule(CAN_MODULE);
}


static void CAN_Module_Disable(void)
{
    /* Disable run */
    CAN_Disable(CAN) ;

    /* FIXME add timeout control */
    while (CAN_GetOperationMode(CAN) != CAN_OPERATE_STOP) {}
}

static void CAN_Gpio_Init(void)
{
    #if defined(SPD1179)
    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_CAN_TXD);
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_CAN_RXD);
    #else
    PIN_SetChannel(PIN_GPIO38, PIN_GPIO38_CAN_TXD);
    PIN_SetChannel(PIN_GPIO39, PIN_GPIO39_CAN_RXD);
    #endif
}

static void CAN_Config(void)
{
    uint32_t u32Data ;
    uint32_t u32DateSegment1;

    double f64NBps ;
    double f64DBps ;

    uint32_t u32Prescaler;

    /* Set IO swap */
    CAN_DisablePinSwap(CAN) ;

    /* Set FDCAN fromat support */
    CAN_DisableFDFormat(CAN);

    /* Set ISO-CAN frame support */
    CAN_DisableNonIsoMode(CAN);

    /* Set protocol exception mode */
    CAN_EnableProtocolException(CAN);

    /* Set restricted mode */
    CAN_DisableRestrictedMode(CAN);

    /* Set monitor mode */
    CAN_DisableMonitorMode(CAN);

    /* Set transmission pause function */
    CAN_EnableTransmissionPause(CAN);

    /* Set RXD edge filter during integration status */
    CAN_EnableEdgeFilter(CAN);

    /* Enable re-tran message afte lost arbitration or error */
    CAN_EnableAutoRetransmission(CAN);

    /* Enable auto start bus-off recovery sequence after bus-off */
    CAN_EnableAutoBusOn(CAN);
```

```
    /* Set nominal bit baud rate
     * 1MHz for 100MHz clock
     **/
    CAN_SetNominalBitRatePrescaler(CAN, 1);
    CAN_SetNominalBitPhaseBufferSegment1(CAN, 79);
    CAN_SetNominalBitPhaseBufferSegment2(CAN, 20); /* sample point: 80% */
    CAN_SetNominalBitResyncJumpWidth(CAN, 8);

    /* Set data bit baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Prescaler = CAN_GetNominalBitRatePrescaler(CAN);
        /* 5Mbps for 100MHz clock */
        CAN_SetDataBitRatePrescaler(CAN, u32Prescaler);
        CAN_SetDataBitPhaseBufferSegment1(CAN, 17 );
        CAN_SetDataBitPhaseBufferSegment2(CAN, 2 ); /* sample point: 90% */
        CAN_SetDataBitResyncJumpWidth    (CAN, 2 );
    }

    /* Calculate nominal phase baud rate */
    u32Data = CAN_GetNominalBitPhaseBufferSegment1(CAN) + 1;
    u32Data += CAN_GetNominalBitPhaseBufferSegment2(CAN);
    u32Data *= CAN_GetNominalBitRatePrescaler(CAN);
    f64NBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;

    printf("[CAN]NBps: %f\n", f64NBps ) ;

    /* Calculate data phase baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Data = CAN_GetDataBitPhaseBufferSegment1(CAN) + 1;
        u32Data += CAN_GetDataBitPhaseBufferSegment2(CAN);
        u32Data *= CAN_GetDataBitRatePrescaler(CAN);
        f64DBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;
        printf("[CAN]DBps: %f\n", f64DBps ) ;
    }

    /* Set delay befor bus-off recovery, based on nominal bit time */
    /* Set delay 1ms */
    u32Data = (uint32_t)(1.0e6 / ( 1.0e9 / f64NBps));
    if (u32Data > 65535)
    {
        u32Data = 65535;
    }

    CAN_SetBusOffRecoveryDelay(CAN, u32Data) ;

    /* Set transmitter delay compensation for FD frame */
    CAN_EnableTransmitterDelayCompensation(CAN);

    /* Set transmitter delay compensation offset and window for FD frame */
    u32DateSegment1 = CAN_GetDataBitPhaseBufferSegment1(CAN);
    CAN_SetTransmitterDelayOffset(CAN, u32DateSegment1 + 1 );
    CAN_SetTransmitterDelayWindow(CAN, u32DateSegment1 + 1 );

    /* Set message RAM parity check */
    CAN_EnableParityCheck(CAN);

    /* Set timestamp */
    CAN_EnableTimestamp(CAN);
}
```

```
static void CAN_Printf_Message(CAN_MessageTypeDef *msg)
{
    int i;

    if (msg->eType == CAN_FRAME_DATA)
    {
        printf("    ") ;
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            printf("0x%02X,", msg->pu8Data[i]);
            if (i % 8 == 7)
            {
                printf("\n    ");
            }
        }
    }

    printf("\n");
}



/*****************************************************************************
 *
 * @brief     In this case, the CAN Transmit Standard and Extended frame data.
 *
 *****************************************************************************/
int main(void)
{
    int i;
    ErrorStatus status;

    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("CAN: Transmit Message\n");

    /* CAN Clk Enable */
    CAN_Clk_Enable();

    /* CAN Disable */
    CAN_Module_Disable();

    /* CAN Gpio Init */
    CAN_Gpio_Init();

    /* CAN config */
    CAN_Config();

    /* Init message RAM contents before enable CAN module */
    CAN_InitMessageRAM(CAN);

    /* CAN run */
    CAN_Enable(CAN);
```

```
    /* Init mailbox for transmit CAN Standard Remote frame message1 */
    message1.eDataDir    = CAN_MSG_DATA_TX;
    message1.eRmtRspEn   = DISABLE;
    message1.eIntEn      = ENABLE;
    message1.eEobEn      = DISABLE;
    message1.eBrs        = DISABLE;
    message1.eEsiCtlEn   = ENABLE;
    message1.eEsi        = DISABLE;

    message1.u32Id       = 0x11;
    message1.u8MBoxId    = 0;
    message1.eFormat     = CAN_FORMAT_STD;
    message1.eType       = CAN_FRAME_REMOTE;

    /* Remote frame no DATA FIELD */
    message1.u8DataLen = 0;
    message1.pu8Data     = NULL;

    status = CAN_SetMessage(CAN, &message1);
    if (status == ERROR)
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message1.u8MBoxId);
        return -1;
    }

    CAN_EnableMailbox(CAN, message1.u8MBoxId);

    /* Wait CAN Standard Remote frame message1 txed */
    while (CAN_GetMailboxControlInfo(CAN, message1.u8MBoxId,
CAN_MBOX_SET_MSG_NEW | CAN_MBOX_REQUEST_TX)) {}

    printf("\nTransmit CAN Standard Remote frame success\n");

    /* Init mailbox for transmit CAN Standard Data frame message2 */
    message2.eDataDir    = CAN_MSG_DATA_TX;
    message2.eRmtRspEn   = DISABLE;
    message2.eIntEn      = ENABLE;
    message2.eEobEn      = DISABLE;
    message2.eBrs        = DISABLE;
    message2.eEsiCtlEn   = ENABLE;
    message2.eEsi        = DISABLE;

    message2.u32Id       = 0x22;
    message2.u8MBoxId    = 1;
    message2.eFormat     = CAN_FORMAT_STD;
    message2.eType       = CAN_FRAME_DATA;

    /* DATA FIELD */
    message2.u8DataLen   = CAN_DATA_LEN;
    message2.pu8Data     = au8TxBuffer;

    for (i = 0; i < message2.u8DataLen; i++)
    {
        message2.pu8Data[i] = i;
    }

    #if DEBUG_INFO
    printf("CAN Standard Frame Data:\n") ;
    CAN_Printf_Message(&message2);
    #endif

    status = CAN_SetMessage(CAN, &message2);
    if (status == ERROR)
```

```
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message2.u8MBoxId);
        return -1;
    }

    CAN_EnableMailbox(CAN, message2.u8MBoxId);

    /* Wait CAN Standard Data frame message2 txed */
    while (CAN_GetMailboxControlInfo(CAN, message2.u8MBoxId,
CAN_MBOX_SET_MSG_NEW | CAN_MBOX_REQUEST_TX)) {}

    printf("\nTransmit CAN Standard Data frame success\n");


    /* Init mailbox for transmit CAN Extended Remote frame message3 */
    message3.eDataDir    = CAN_MSG_DATA_TX;
    message3.eRmtRspEn   = DISABLE;
    message3.eIntEn      = ENABLE;
    message3.eEobEn      = DISABLE;
    message3.eBrs        = DISABLE;
    message3.eEsiCtlEn   = ENABLE;
    message3.eEsi        = DISABLE;

    message3.u32Id       = 0x33;
    message3.u8MBoxId    = 3;
    message3.eFormat     = CAN_FORMAT_EXT;
    message3.u8DataLen   = CAN_DATA_LEN;
    message3.eType       = CAN_FRAME_REMOTE;

    /* Remote frame no DATA FIELD */
    message3.u8DataLen   = 0;
    message3.pu8Data     = NULL;

    status = CAN_SetMessage(CAN, &message3);
    if (status == ERROR)
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message2.u8MBoxId);
        return -1;
    }

    CAN_EnableMailbox(CAN, message3.u8MBoxId);

    /* Wait CAN Extended Remote frame message3 txed */
    while (CAN_GetMailboxControlInfo(CAN, message3.u8MBoxId,
CAN_MBOX_SET_MSG_NEW | CAN_MBOX_REQUEST_TX)) {}

    printf("\nTransmit CAN Extended Remote frame success\n");


    /* Init mailbox for transmit CAN Extended Data frame message4 */
    message4.eDataDir    = CAN_MSG_DATA_TX;
    message4.eRmtRspEn   = DISABLE;
    message4.eIntEn      = ENABLE;
    message4.eEobEn      = DISABLE;
    message4.eBrs        = DISABLE;
    message4.eEsiCtlEn   = ENABLE;
    message4.eEsi        = DISABLE;

    message4.u32Id       = 0x44;
    message4.u8MBoxId    = 4;
    message4.eFormat     = CAN_FORMAT_EXT;
    message4.eType       = CAN_FRAME_DATA;
```

```
    /* DATA FIELD */
    message4.u8DataLen   = CAN_DATA_LEN;
    message4.pu8Data     = au8TxBuffer;

    for (i = 0; i < message4.u8DataLen; i++)
    {
        message4.pu8Data[i] = i;
    }

    #if DEBUG_INFO
    printf("CAN Extended Frame Data:\n") ;
    CAN_Printf_Message(&message4);
    #endif

    status = CAN_SetMessage(CAN, &message4);
    if (status == ERROR)
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message4.u8MBoxId);
        return -1;
    }

    CAN_EnableMailbox(CAN, message4.u8MBoxId);

    /* Wait CAN Extended Data frame message4 txed */
    while (CAN_GetMailboxControlInfo(CAN, message4.u8MBoxId,
CAN_MBOX_SET_MSG_NEW | CAN_MBOX_REQUEST_TX)) {}

    printf("\nTransmit CAN Extended Data frame success\n");

    while(1)
    {

    }
}
```

7.1.2 接收标准帧数据

本示例中演示使用 CAN 协议依次接收标准格式远程帧、标准格式数据帧信息，传输的速率为 1Mbps，采样点为 80%，其配置流程如下：

- 调用 CAN_DisableFDFormat()函数失能 CANFD 协议，使用默认的典型 CAN 协议；
- 调用 CAN_DisableNonIsoMode()函数失能 NONISO 模式，使用默认的 ISO 模式；
- 调用 CAN_SetNominalBitRatePrescaler()、CAN_SetNominalBitPhaseBufferSegment1()、CAN_SetNominalBitPhaseBufferSegment2()函数，根据 6.2 章节计算 CAN 的通信速率和采样点；
- 配置接收标准格式的远程帧数据；
  eDataDir= CAN_MSG_DATA_RX 配置消息传输的方向为接收；
  u32Id = 0x11 配置传输消息的标识符为 0x11；
  u32IdMask = 0xff 配置传输消息标识符屏蔽位为 0xff；
  u8MBoxId = 0 配置使用的邮箱的 ID 为 0；
  eFormat = CAN_FORMAT_STD 配置传输的消息为标准格式；
  eType = CAN_FRAME_REMOTE 配置传输的消息为远程帧类型；

u8DataLen = 0 配置传输数据的长度为 0；
pu8Data = NULL 配置传输的数据为空；

- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；
- 配置接收标准格式的数据帧数据；
  eDataDir= CAN_MSG_DATA_RX 配置消息传输的方向为接收；
  u32Id = 0x22 配置传输消息的标识符为 0x22；
  u32IdMask = 0xff 配置传输消息标识符屏蔽位为 0xff；
  u8MBoxId = 0 配置使用的邮箱的 ID 为 1；
  eFormat = CAN_FORMAT_STD 配置传输的消息为标准格式；
  eType = CAN_FRAME_DATA 配置传输的消息为数据帧类型；
  u8DataLen = CAN_DATA_LEN 配置传输数据的长度为 CAN_DATA_LEN；
  pu8Data = (uint8_t*)(long)(&CAN->CANMBOX[message.u8MBoxId].CANMBOXFDW[0]) 获取传输的数据；
- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；

接收标准帧数据

```
#include <stdio.h>
#include <stdlib.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


#define DEBUG_INFO     1

#define CAN_DATA_LEN       8

/* CAN Standard Remote frame message1 */
CAN_MessageTypeDef message1;
/* CAN Standard Data frame message2 */
CAN_MessageTypeDef message2;

static void CAN_Clk_Enable(void)
{
    /* CAN Clk Enable */
    CLOCK_SetModuleDiv(CAN_MODULE, 1);
    CLOCK_EnableModule(CAN_MODULE);
}


static void CAN_Module_Disable(void)
{
    /* Disable run */
    CAN_Disable(CAN) ;

    /* FIXME add timeout control */
    while (CAN_GetOperationMode(CAN) != CAN_OPERATE_STOP) {}
}

static void CAN_Gpio_Init(void)
{
    #if defined(SPD1179)
```

```
    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_CAN_TXD);
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_CAN_RXD);
    #else
    PIN_SetChannel(PIN_GPIO38, PIN_GPIO38_CAN_TXD);
    PIN_SetChannel(PIN_GPIO39, PIN_GPIO39_CAN_RXD);
    #endif
}

static void CAN_Config(void)
{
    uint32_t u32Data ;
    uint32_t u32DateSegment1 ;

    double f64NBps ;
    double f64DBps ;

    uint32_t u32Prescaler;

    /* Set IO swap */
    CAN_DisablePinSwap(CAN) ;

    /* Set FDCAN fromat support */
    CAN_DisableFDFormat(CAN);

    /* Set ISO-CAN frame support */
    CAN_DisableNonIsoMode(CAN);

    /* Set protocol exception mode */
    CAN_EnableProtocolException(CAN);

    /* Set restricted mode */
    CAN_DisableRestrictedMode(CAN);

    /* Set monitor mode */
    CAN_DisableMonitorMode(CAN);

    /* Set transmission pause function */
    CAN_EnableTransmissionPause(CAN);

    /* Set RXD edge filter during integration status */
    CAN_EnableEdgeFilter(CAN);

    /* Enable re-tran message afte lost arbitration or error */
    CAN_EnableAutoRetransmission(CAN);

    /* Enable auto start bus-off recovery sequence after bus-off */
    CAN_EnableAutoBusOn(CAN);

    /* Set nominal bit baud rate
     * 1MHz for 100MHz clock
     **/
    CAN_SetNominalBitRatePrescaler(CAN, 1);
    CAN_SetNominalBitPhaseBufferSegment1(CAN, 79);
    CAN_SetNominalBitPhaseBufferSegment2(CAN, 20); /* sample point: 80% */
    CAN_SetNominalBitResyncJumpWidth(CAN, 8);

    /* Set data bit baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Prescaler = CAN_GetNominalBitRatePrescaler(CAN);
        /* 5Mbps for 100MHz clock */
        CAN_SetDataBitRatePrescaler(CAN, u32Prescaler);
        CAN_SetDataBitPhaseBufferSegment1(CAN, 17 );
```

```
        CAN_SetDataBitPhaseBufferSegment2(CAN, 2 ); /* sample point: 90% */
        CAN_SetDataBitResyncJumpWidth    (CAN, 2 );
    }

    /* Calculate nominal phase baud rate */
    u32Data = CAN_GetNominalBitPhaseBufferSegment1(CAN) + 1;
    u32Data += CAN_GetNominalBitPhaseBufferSegment2(CAN);
    u32Data *= CAN_GetNominalBitRatePrescaler(CAN);
    f64NBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;

    printf("[CAN]NBps: %f\n", f64NBps ) ;

    /* Calculate data phase baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Data = CAN_GetDataBitPhaseBufferSegment1(CAN) + 1;
        u32Data += CAN_GetDataBitPhaseBufferSegment2(CAN);
        u32Data *= CAN_GetDataBitRatePrescaler(CAN);
        f64DBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;
        printf("[CAN]DBps: %f\n", f64DBps ) ;
    }

    /* Set delay befor bus-off recovery, based on nominal bit time */
    /* Set delay 1ms */
    u32Data = (uint32_t)(1.0e6 / ( 1.0e9 / f64NBps));
    if (u32Data > 65535)
    {
        u32Data = 65535;
    }

    CAN_SetBusOffRecoveryDelay(CAN, u32Data) ;

    /* Set transmitter delay compensation for FD frame */
    CAN_EnableTransmitterDelayCompensation(CAN);

    /* Set transmitter delay compensation offset and window for FD frame */
    u32DateSegment1 = CAN_GetDataBitPhaseBufferSegment1(CAN);
    CAN_SetTransmitterDelayOffset(CAN, u32DateSegment1 + 1 );
    CAN_SetTransmitterDelayWindow(CAN, u32DateSegment1 + 1 );

    /* Set message RAM parity check */
    CAN_EnableParityCheck(CAN);

    /* Set timestamp */
    CAN_EnableTimestamp(CAN);
}


static void CAN_Printf_Message(CAN_MessageTypeDef *msg)
{
    int i;

    if (msg->eType == CAN_FRAME_DATA)
    {
        printf("    ") ;
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            printf("0x%02X,", msg->pu8Data[i]);
            if (i % 8 == 7)
            {
                printf("\n    ");
            }
        }
```

```
    }

    printf("\n");
}

static ErrorStatus CAN_CheckMessageData(CAN_MessageTypeDef *msg)
{
    int i;
    ErrorStatus status = SUCCESS;

    if (msg->eType == CAN_FRAME_DATA)
    {
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            if (msg->pu8Data[i] != i)
            {
                printf("Data Error: pu8Data[%d] != %d\n", i, i);
                status = ERROR;
            }
        }
    }

    return status;
}



/*****************************************************************************
*
 *
 * @brief      In this case, the CAN receive Standard frame data.
 *

*****************************************************************************/
int main(void)
{
    ErrorStatus status;

    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("CAN: Receive Message\n");

    /* CAN Clk Enable */
    CAN_Clk_Enable();

    /* CAN Disable */
    CAN_Module_Disable();

    /* CAN Gpio Init */
    CAN_Gpio_Init();

    /* CAN config */
    CAN_Config();
```

```
    /* Init message RAM contents before enable CAN module */
    CAN_InitMessageRAM(CAN);

    /* CAN run */
    CAN_Enable(CAN);

    /* Init mailbox for receive message1 */
    message1.eDataDir    = CAN_MSG_DATA_RX;
    message1.eRmtRspEn   = DISABLE;
    message1.eIntEn      = ENABLE;
    message1.eEobEn      = DISABLE;
    message1.eOverwriteEn= ENABLE;
    message1.eMaskRtrEn  = ENABLE;
    message1.eMaskIdeEn  = ENABLE;

    message1.u32Id       = 0x11;
    message1.u32IdMask   = 0xff;
    message1.u8MBoxId    = 0;
    message1.eFormat     = CAN_FORMAT_STD;
    message1.eType       = CAN_FRAME_REMOTE;

    /* Remote frame no DATA FIELD */
    message1.pu8Data     = NULL;
    message1.u8DataLen   = 0;

    status = CAN_SetMessage(CAN, &message1);
    if (status == ERROR)
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message1.u8MBoxId);
        return -1;
    }

    CAN_EnableMailbox(CAN, message1.u8MBoxId);

    /* Wait receive CAN Standard Remote frame message1 */
    while (!CAN_GetMailboxControlInfo(CAN, message1.u8MBoxId,
CAN_MBOX_SET_MSG_NEW)) {}

    CAN_GetMessage( CAN, &message1);

    printf("\nReceives CAN Standard Remote frame success\n");


    /* Init mailbox for receive message2 */
    message2.eDataDir    = CAN_MSG_DATA_RX;
    message2.eRmtRspEn   = DISABLE;
    message2.eIntEn      = ENABLE;
    message2.eEobEn      = DISABLE;
    message2.eOverwriteEn= ENABLE;
    message2.eMaskRtrEn  = ENABLE;
    message2.eMaskIdeEn  = ENABLE;

    message2.u32Id       = 0x22;
    message2.u32IdMask   = 0xff;
    message2.u8MBoxId    = 1;
    message2.eFormat     = CAN_FORMAT_STD;
    message2.eType       = CAN_FRAME_DATA;

    /* DATA FIELD */
    message2.u8DataLen   = CAN_DATA_LEN;

    status = CAN_SetMessage(CAN, &message2);
    if (status == ERROR)
```

```
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message2.u8MBoxId);
        return -1;
    }

    message2.pu8Data = (uint8_t *)(long)(& CAN->CANMBOX[message2.u8MBoxId].CANMBOXFDW[0]);

    CAN_EnableMailbox(CAN, message2.u8MBoxId);

    /* Wait receive CAN Standard Data frame message2 */
    while (!CAN_GetMailboxControlInfo(CAN, message2.u8MBoxId, CAN_MBOX_SET_MSG_NEW)) {}

    CAN_GetMessage( CAN, &message1);

    #if DEBUG_INFO
    printf("CAN Standard Frame Data:\n") ;
    CAN_Printf_Message(&message2);
    #endif

    status = CAN_CheckMessageData(&message2);
    if (status != SUCCESS)
    {
        printf("Receives CAN Standard Data frame fail\n");
    }

    printf("Receives CAN Standard Data frame success\n");

    while(1)
    {

    }
}
```

7.1.3 接收扩展帧数据

本示例中演示使用 CAN 协议依次接收扩展格式远程帧和扩展格式数据帧信息，传输的速率为 1Mbps，采样点为 80%，其配置流程如下：

- 调用 CAN_DisableFDFormat()函数失能 CANFD 协议，使用默认的典型 CAN 协议；
- 调用 CAN_DisableNonIsoMode()函数失能 NONISO 模式，使用默认的 ISO 模式；
- 调用 CAN_SetNominalBitRatePrescaler()、CAN_SetNominalBitPhaseBufferSegment1()、CAN_SetNominalBitPhaseBufferSegment2()函数，根据 6.2 章节计算 CAN 的通信速率和采样点；
- 配置接收扩展格式的远程帧数据；
  eDataDir= CAN_MSG_DATA_RX 配置消息传输的方向为接收；
  u32Id = 0x33 配置传输消息的标识符为 0x33；
  u32IdMask = 0xff 配置传输消息标识符屏蔽位为 0xff；
  u8MBoxId = 0 配置使用的邮箱的 ID 为 0；
  eFormat = CAN_FORMAT_EXT 配置传输的消息为扩展格式；

eType = CAN_FRAME_REMOTE 配置传输的消息为远程帧类型；
u8DataLen = 0 配置传输数据的长度为 0；
pu8Data = NULL 配置传输的数据为空；

- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；
- 配置接收标准格式的数据帧数据；
  eDataDir= CAN_MSG_DATA_RX 配置消息传输的方向为接收；
  u32Id = 0x44 配置传输消息的标识符为 0x44；
  u32IdMask = 0xff 配置传输消息标识符屏蔽位为 0xff；
  u8MBoxId = 1 配置使用的邮箱的 ID 为 1；
  eFormat = CAN_FORMAT_EXT 配置传输的消息为扩展格式；
  eType = CAN_FRAME_DATA 配置传输的消息为数据帧类型；
  u8DataLen = CAN_DATA_LEN 配置传输数据的长度为 CAN_DATA_LEN；
  pu8Data = (uint8_t*)(long)(&CAN->CANMBOX[message.u8MBoxId].CANMBOXFDW[0]) 获取传输的数据；
- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；

接收扩展帧数据

```
#include <stdio.h>
#include <stdlib.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


#define DEBUG_INFO     1

#define CAN_DATA_LEN       8

/* CAN Extended Remote frame message1 */
CAN_MessageTypeDef message1;
/* CAN Extended Data frame message2 */
CAN_MessageTypeDef message2;

static void CAN_Clk_Enable(void)
{
    /* CAN Clk Enable */
    CLOCK_SetModuleDiv(CAN_MODULE, 1);
    CLOCK_EnableModule(CAN_MODULE);
}


static void CAN_Module_Disable(void)
{
    /* Disable run */
    CAN_Disable(CAN) ;

    /* FIXME add timeout control */
    while (CAN_GetOperationMode(CAN) != CAN_OPERATE_STOP) {}
}

static void CAN_Gpio_Init(void)
```

```
{
    #if defined(SPD1179)
    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_CAN_TXD);
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_CAN_RXD);
    #else
    PIN_SetChannel(PIN_GPIO38, PIN_GPIO38_CAN_TXD);
    PIN_SetChannel(PIN_GPIO39, PIN_GPIO39_CAN_RXD);
    #endif
}

static void CAN_Config(void)
{
    uint32_t u32Data ;
    uint32_t u32DateSegment1 ;

    double f64NBps ;
    double f64DBps ;

    uint32_t u32Prescaler;

    /* Set IO swap */
    CAN_DisablePinSwap(CAN) ;

    /* Set FDCAN fromat support */
    CAN_DisableFDFormat(CAN);

    /* Set ISO-CAN frame support */
    CAN_DisableNonIsoMode(CAN);

    /* Set protocol exception mode */
    CAN_EnableProtocolException(CAN);

    /* Set restricted mode */
    CAN_DisableRestrictedMode(CAN);

    /* Set monitor mode */
    CAN_DisableMonitorMode(CAN);

    /* Set transmission pause function */
    CAN_EnableTransmissionPause(CAN);

    /* Set RXD edge filter during integration status */
    CAN_EnableEdgeFilter(CAN);

    /* Enable re-tran message afte lost arbitration or error */
    CAN_EnableAutoRetransmission(CAN);

    /* Enable auto start bus-off recovery sequence after bus-off */
    CAN_EnableAutoBusOn(CAN);

    /* Set nominal bit baud rate
     * 1MHz for 100MHz clock
     **/
    CAN_SetNominalBitRatePrescaler(CAN, 1);
    CAN_SetNominalBitPhaseBufferSegment1(CAN, 79);
    CAN_SetNominalBitPhaseBufferSegment2(CAN, 20); /* sample point: 80% */
    CAN_SetNominalBitResyncJumpWidth(CAN, 8);

    /* Set data bit baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Prescaler = CAN_GetNominalBitRatePrescaler(CAN);
        /* 5Mbps for 100MHz clock */
```

```
        CAN_SetDataBitRatePrescaler(CAN, u32Prescaler);
        CAN_SetDataBitPhaseBufferSegment1(CAN, 17 );
        CAN_SetDataBitPhaseBufferSegment2(CAN, 2 ); /* sample point: 90% */
        CAN_SetDataBitResyncJumpWidth    (CAN, 2 );
    }

    /* Calculate nominal phase baud rate */
    u32Data = CAN_GetNominalBitPhaseBufferSegment1(CAN) + 1;
    u32Data += CAN_GetNominalBitPhaseBufferSegment2(CAN);
    u32Data *= CAN_GetNominalBitRatePrescaler(CAN);
    f64NBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;

    printf("[CAN]NBps: %f\n", f64NBps ) ;

    /* Calculate data phase baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Data = CAN_GetDataBitPhaseBufferSegment1(CAN) + 1;
        u32Data += CAN_GetDataBitPhaseBufferSegment2(CAN);
        u32Data *= CAN_GetDataBitRatePrescaler(CAN);
        f64DBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;
        printf("[CAN]DBps: %f\n", f64DBps ) ;
    }

    /* Set delay befor bus-off recovery, based on nominal bit time */
    /* Set delay 1ms */
    u32Data = (uint32_t)(1.0e6 / ( 1.0e9 / f64NBps));
    if (u32Data > 65535)
    {
        u32Data = 65535;
    }

    CAN_SetBusOffRecoveryDelay(CAN, u32Data) ;

    /* Set transmitter delay compensation for FD frame */
    CAN_EnableTransmitterDelayCompensation(CAN);

    /* Set transmitter delay compensation offset and window for FD frame */
    u32DateSegment1 = CAN_GetDataBitPhaseBufferSegment1(CAN);
    CAN_SetTransmitterDelayOffset(CAN, u32DateSegment1 + 1 );
    CAN_SetTransmitterDelayWindow(CAN, u32DateSegment1 + 1 );

    /* Set message RAM parity check */
    CAN_EnableParityCheck(CAN);

    /* Set timestamp */
    CAN_EnableTimestamp(CAN);
}


static void CAN_Printf_Message(CAN_MessageTypeDef *msg)
{
    int i;

    if (msg->eType == CAN_FRAME_DATA)
    {
        printf("    ") ;
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            printf("0x%02X,", msg->pu8Data[i]);
            if (i % 8 == 7)
            {
                printf("\n    ");
```

```
            }
        }
    }

    printf("\n");
}

static ErrorStatus CAN_CheckMessageData(CAN_MessageTypeDef *msg)
{
    int i;
    ErrorStatus status = SUCCESS;

    if (msg->eType == CAN_FRAME_DATA)
    {
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            if (msg->pu8Data[i] != i)
            {
                printf("Data Error: pu8Data[%d] != %d\n", i, i);
                status = ERROR;
            }
        }
    }

    return status;
}



/*****************************************************************************
 *
 * @brief     In this case, the CAN Receive Extended frame data.
 *
 *****************************************************************************/
int main(void)
{
    ErrorStatus status;

    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("CAN: Receive Message\n");

    /* CAN Clk Enable */
    CAN_Clk_Enable();

    /* CAN Disable */
    CAN_Module_Disable();

    /* CAN Gpio Init */
    CAN_Gpio_Init();

    /* CAN config */
    CAN_Config();
```

```
    /* Init message RAM contents before enable CAN module */
    CAN_InitMessageRAM(CAN);

    /* CAN run */
    CAN_Enable(CAN);

    /* Init mailbox for receive message1 */
    message1.eDataDir    = CAN_MSG_DATA_RX;
    message1.eRmtRspEn   = DISABLE;
    message1.eIntEn      = ENABLE;
    message1.eEobEn      = DISABLE;
    message1.eOverwriteEn= ENABLE;
    message1.eMaskRtrEn  = ENABLE;
    message1.eMaskIdeEn  = ENABLE;

    message1.u32Id       = 0x33;
    message1.u32IdMask   = 0xff;
    message1.u8MBoxId    = 0;
    message1.eFormat     = CAN_FORMAT_EXT;
    message1.eType       = CAN_FRAME_REMOTE;

    /* Remote frame no DATA FIELD */
    message1.pu8Data     = NULL;
    message1.u8DataLen   = 0;

    status = CAN_SetMessage(CAN, &message1);
    if (status == ERROR)
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message1.u8MBoxId);
        return -1;
    }

    CAN_EnableMailbox(CAN, message1.u8MBoxId);

    /* Wait Receive CAN Extended Remote frame message1 */
    while (!CAN_GetMailboxControlInfo(CAN, message1.u8MBoxId,
CAN_MBOX_SET_MSG_NEW)) {}

    CAN_GetMessage( CAN, &message1);

    printf("\nReceives CAN Extended Remote frame success\n");


    /* Init mailbox for receive message2 */
    message2.eDataDir    = CAN_MSG_DATA_RX;
    message2.eRmtRspEn   = DISABLE;
    message2.eIntEn      = ENABLE;
    message2.eEobEn      = DISABLE;
    message2.eOverwriteEn= ENABLE;
    message2.eMaskRtrEn  = ENABLE;
    message2.eMaskIdeEn  = ENABLE;

    message2.u32Id       = 0x44;
    message2.u32IdMask   = 0xff;
    message2.u8MBoxId    = 1;
    message2.eFormat     = CAN_FORMAT_EXT;
    message2.eType       = CAN_FRAME_DATA;

    /* DATA FIELD */
    message2.u8DataLen   = CAN_DATA_LEN;

    status = CAN_SetMessage(CAN, &message2);
    if (status == ERROR)
```

```
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message2.u8MBoxId);
        return -1;
    }

    message2.pu8Data = (uint8_t *)(long)(& CAN-
>CANMBOX[message2.u8MBoxId].CANMBOXFDW[0]);

    CAN_EnableMailbox(CAN, message2.u8MBoxId);

    /* Wait receive CAN Extended Data frame message2 */
    while (!CAN_GetMailboxControlInfo(CAN, message2.u8MBoxId,
CAN_MBOX_SET_MSG_NEW)) {}

    CAN_GetMessage( CAN, &message1);

    #if DEBUG_INFO
    printf("CAN Extended Frame Data:\n") ;
    CAN_Printf_Message(&message2);
    #endif

    status = CAN_CheckMessageData(&message2);
    if (status != SUCCESS)
    {
        printf("Receives CAN Extended Data frame fail\n");
    }

    printf("Receives CAN Extended Data frame success\n");

    while(1)
    {

    }
}
```

7.2 CANFD 协议传输

7.2.1 发送标准帧和扩展帧数据

本示例中演示使用 CANFD 协议依次发送标准格式数据帧和扩展格式数据帧信息，慢速阶段的传输速率为 1Mbps，采样点为 80%，快速阶段的传输速率为 5Mbps，采样点为 90%，其配置流程如下：

- 调用 CAN_EnableFDFormat ()函数使能 CANFD 协议；
- 调用 CAN_DisableNonIsoMode()函数失能 NONISO 模式，使用默认的 ISO 模式；
- 调用 CAN_SetNominalBitRatePrescaler()、CAN_SetNominalBitPhaseBufferSegment1()、CAN_SetNominalBitPhaseBufferSegment2()函数，根据 6.2 章节计算 CAN 的通信速率和采样点；
- 配置发送标准格式的数据帧数据；
  eDataDir= CAN_MSG_DATA_TX 配置消息传输的方向为发送；
  u32Id = 0x22 配置传输消息的标识符为 0x22；
  u8MBoxId = 1 配置使用的邮箱的 ID 为 1；
  eFormat = CAN_FORMAT_FD_STD 配置传输的消息为标准 FD 格式；
  eType = CAN_FRAME_DATA 配置传输的消息为数据帧类型；
  u8DataLen = CANFD_DATA_LEN 配置传输数据的长度为 CANFD_DATA_LEN；
  pu8Data = au8TxBuffer 配置传输的数据为 au8TxBuffer；
- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；
- 配置发送扩展格式的数据帧数据；
  eDataDir= CAN_MSG_DATA_TX 配置消息传输的方向为发送；
  u32Id = 0x44 配置传输消息的标识符为 0x44；
  u8MBoxId = 4 配置使用的邮箱的 ID 为 4；
  eFormat = CAN_FORMAT_FD_EXT 配置传输的消息为扩展 FD 格式；
  eType = CAN_FRAME_DATAE 配置传输的消息为数据帧类型；
  u8DataLen = CAN_DATA_LEN 配置传输数据的长度为 CAN_DATA_LEN；
  pu8Data = au8TxBuffer 配置传输的数据为 au8TxBuffer；
- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；

发送标准帧和扩展帧数据

```
#include <stdio.h>
#include <stdlib.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif

#define DEBUG_INFO      1
#define CANFD_DATA_LEN      64

uint8_t au8TxBuffer[64];
```

```
/* CANFD Standard Data frame message1 */
CAN_MessageTypeDef message1;
/* CANFD Extended Data frame message2 */
CAN_MessageTypeDef message2;

static void CAN_Clk_Enable(void)
{
    /* CAN Clk Enable */
    CLOCK_SetModuleDiv(CAN_MODULE, 1);
    CLOCK_EnableModule(CAN_MODULE);
}


static void CAN_Module_Disable(void)
{
    /* Disable run */
    CAN_Disable(CAN) ;

    /* FIXME add timeout control */
    while (CAN_GetOperationMode(CAN) != CAN_OPERATE_STOP) {}
}

static void CAN_Gpio_Init(void)
{
    #if defined(SPD1179)
    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_CAN_TXD);
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_CAN_RXD);
    #else
    PIN_SetChannel(PIN_GPIO38, PIN_GPIO38_CAN_TXD);
    PIN_SetChannel(PIN_GPIO39, PIN_GPIO39_CAN_RXD);
    #endif
}

static void CAN_Config(void)
{
    uint32_t u32Data ;
    uint32_t u32DateSegment1 ;

    double f64NBps ;
    double f64DBps ;

    uint32_t u32Prescaler;

    /* Set IO swap */
    CAN_DisablePinSwap(CAN) ;

    /* Set FDCAN fromat support */
    CAN_EnableFDFormat(CAN);

    /* Set ISO-CAN frame support */
    CAN_DisableNonIsoMode(CAN);

    /* Set protocol exception mode */
    CAN_EnableProtocolException(CAN);

    /* Set restricted mode */
    CAN_DisableRestrictedMode(CAN);

    /* Set monitor mode */
    CAN_DisableMonitorMode(CAN);

    /* Set transmission pause function */
```

```
    CAN_EnableTransmissionPause(CAN);

    /* Set RXD edge filter during integration status */
    CAN_EnableEdgeFilter(CAN);

    /* Enable re-tran message afte lost arbitration or error */
    CAN_EnableAutoRetransmission(CAN);

    /* Enable auto start bus-off recovery sequence after bus-off */
    CAN_EnableAutoBusOn(CAN);

    /* Set nominal bit baud rate
     * 1MHz for 100MHz clock
     **/
    CAN_SetNominalBitRatePrescaler(CAN, 1);
    CAN_SetNominalBitPhaseBufferSegment1(CAN, 79);
    CAN_SetNominalBitPhaseBufferSegment2(CAN, 20); /* sample point: 80% */
    CAN_SetNominalBitResyncJumpWidth(CAN, 8);

    /* Set data bit baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Prescaler = CAN_GetNominalBitRatePrescaler(CAN);
        /* 5Mbps for 100MHz clock */
        CAN_SetDataBitRatePrescaler(CAN, u32Prescaler);
        CAN_SetDataBitPhaseBufferSegment1(CAN, 17 );
        CAN_SetDataBitPhaseBufferSegment2(CAN, 2 ); /* sample point: 90% */
        CAN_SetDataBitResyncJumpWidth    (CAN, 2 );
    }

    /* Calculate nominal phase baud rate */
    u32Data = CAN_GetNominalBitPhaseBufferSegment1(CAN) + 1;
    u32Data += CAN_GetNominalBitPhaseBufferSegment2(CAN);
    u32Data *= CAN_GetNominalBitRatePrescaler(CAN);
    f64NBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;

    printf("[CAN]NBps: %f\n", f64NBps ) ;

    /* Calculate data phase baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Data = CAN_GetDataBitPhaseBufferSegment1(CAN) + 1;
        u32Data += CAN_GetDataBitPhaseBufferSegment2(CAN);
        u32Data *= CAN_GetDataBitRatePrescaler(CAN);
        f64DBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;
        printf("[CAN]DBps: %f\n", f64DBps ) ;
    }

    /* Set delay befor bus-off recovery, based on nominal bit time */
    /* Set delay 1ms */
    u32Data = (uint32_t)(1.0e6 / ( 1.0e9 / f64NBps));
    if (u32Data > 65535)
    {
        u32Data = 65535;
    }

    CAN_SetBusOffRecoveryDelay(CAN, u32Data) ;

    /* Set transmitter delay compensation for FD frame */
    CAN_EnableTransmitterDelayCompensation(CAN);

    /* Set transmitter delay compensation offset and window for FD frame */
    u32DateSegment1 = CAN_GetDataBitPhaseBufferSegment1(CAN);
```

```
    CAN_SetTransmitterDelayOffset(CAN, u32DateSegment1 + 1 );
    CAN_SetTransmitterDelayWindow(CAN, u32DateSegment1 + 1 );

    /* Set message RAM parity check */
    CAN_EnableParityCheck(CAN);

    /* Set timestamp */
    CAN_EnableTimestamp(CAN);
}

static void CAN_Printf_Message(CAN_MessageTypeDef *msg)
{
    int i;

    if (msg->eType == CAN_FRAME_DATA)
    {
        printf("    ") ;
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            printf("0x%02X,", msg->pu8Data[i]);
            if (i % 8 == 7)
            {
                printf("\n    ");
            }
        }
    }

    printf("\n");
}



/*****************************************************************************
 *
 * @brief     In this case, the CANFD Transmit Standard and Extended frame data.
 *
 *****************************************************************************/
int main(void)
{
    int i;
    ErrorStatus status;

    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("CAN: Transmit Message\n");

    /* CAN Clk Enable */
    CAN_Clk_Enable();

    /* CAN Disable */
    CAN_Module_Disable();

    /* CAN Gpio Init */
    CAN_Gpio_Init();
```

```
    /* CAN config */
    CAN_Config();

    /* Init message RAM contents before enable CAN module */
    CAN_InitMessageRAM(CAN);

    /* CAN run */
    CAN_Enable(CAN);

    /* Init mailbox for transmit CANFD Standard Data frame message1 */
    message1.eDataDir    = CAN_MSG_DATA_TX;
    message1.eRmtRspEn   = DISABLE;
    message1.eIntEn      = ENABLE;
    message1.eEobEn      = DISABLE;
    message1.eBrs        = DISABLE;
    message1.eEsiCtlEn   = ENABLE;
    message1.eEsi        = DISABLE;

    message1.u32Id       = 0x22;
    message1.u8MBoxId    = 1;
    message1.eFormat     = CAN_FORMAT_FD_STD;
    message1.eType       = CAN_FRAME_DATA;

    /* DATA FIELD */
    message1.u8DataLen   = CANFD_DATA_LEN;
    message1.pu8Data     = au8TxBuffer;

    for (i = 0; i < message1.u8DataLen; i++)
    {
        message1.pu8Data[i] = i;
    }

    #if DEBUG_INFO
    printf("CANFD Standard Frame Data:\n") ;
    CAN_Printf_Message(&message1);
    #endif

    status = CAN_SetMessage(CAN, &message1);
    if (status == ERROR)
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message1.u8MBoxId);
        return -1;
    }

    CAN_EnableMailbox(CAN, message1.u8MBoxId);

    /* Wait CANFD Standard Data frame message1 txed */
    while (CAN_GetMailboxControlInfo(CAN, message1.u8MBoxId,
CAN_MBOX_SET_MSG_NEW | CAN_MBOX_REQUEST_TX)) {}

    printf("\nTransmit CANFD Standard Data frame success\n");

    /* Init mailbox for transmit CANFD Extended Data frame message2 */
    message2.eDataDir    = CAN_MSG_DATA_TX;
    message2.eRmtRspEn   = DISABLE;
    message2.eIntEn      = ENABLE;
    message2.eEobEn      = DISABLE;
    message2.eBrs        = DISABLE;
    message2.eEsiCtlEn   = ENABLE;
    message2.eEsi        = DISABLE;

    message2.u32Id       = 0x44;
```

```
    message2.u8MBoxId    = 4;
    message2.eFormat     = CAN_FORMAT_FD_EXT;
    message2.eType       = CAN_FRAME_DATA;

    /* DATA FIELD */
    message2.u8DataLen   = CANFD_DATA_LEN;
    message2.pu8Data     = au8TxBuffer;

    for (i = 0; i < message2.u8DataLen; i++)
    {
        message2.pu8Data[i] = i;
    }

    #if DEBUG_INFO
    printf("CANFD Extended Frame Data:\n") ;
    CAN_Printf_Message(&message2);
    #endif

    status = CAN_SetMessage(CAN, &message2);
    if (status == ERROR)
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message2.u8MBoxId);
        return -1;
    }

    CAN_EnableMailbox(CAN, message2.u8MBoxId);

    /* Wait CANFD Extended Data frame message2 txed */
    while (CAN_GetMailboxControlInfo(CAN, message2.u8MBoxId,
CAN_MBOX_SET_MSG_NEW | CAN_MBOX_REQUEST_TX)) {}

    printf("\nTransmit CANFD Extended Data frame success\n");

    while(1)
    {

    }
}
```

7.2.2 接收标准帧数据

本示例中演示使用 CANFD 协议接收标准格式数据帧信息，慢速阶段的传输速率为 1Mbps，采样点为 80%，快速阶段的传输速率为 5Mbps，采样点为 90%，其配置流程如下：

- 调用 CAN_DisableFDFormat()函数使能 CANFD 协议；
- 调用 CAN_DisableNonIsoMode()函数失能 NONISO 模式，使用默认的 ISO 模式；
- 调用 CAN_SetNominalBitRatePrescaler()、CAN_SetNominalBitPhaseBufferSegment1()、CAN_SetNominalBitPhaseBufferSegment2()函数，根据 6.2 章节计算 CAN 的通信速率和采样点；
- 配置接收标准格式的数据帧数据；
  eDataDir= CAN_MSG_DATA_RX 配置消息传输的方向为接收；
  u32Id = 0x22 配置传输消息的标识符为 0x22；
  u32IdMask = 0xff 配置传输消息标识符屏蔽位为 0xff；

u8MBoxId = 1 配置使用的邮箱的 ID 为 1；
eFormat = CAN_FORMAT_FD_STD 配置传输的消息为标准 FD 格式；
eType = CAN_FRAME_DATA 配置传输的消息为数据帧类型；
u8DataLen = CANFD_DATA_LEN 配置传输数据的长度为 CANFD_DATA_LEN；
pu8Data = (uint8_t*)(long)(&CAN->CANMBOX[message.u8MBoxId].CANMBOXFDW[0]) 获取传输的数据；

- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；

接收标准帧数据

```
#include <stdio.h>
#include <stdlib.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


#define DEBUG_INFO      1

#define CANFD_DATA_LEN        64

/* CANFD Standard Data frame message1 */
CAN_MessageTypeDef message1;

static void CAN_Clk_Enable(void)
{
    /* CAN Clk Enable */
    CLOCK_SetModuleDiv(CAN_MODULE, 1);
    CLOCK_EnableModule(CAN_MODULE);
}


static void CAN_Module_Disable(void)
{
    /* Disable run */
    CAN_Disable(CAN) ;

    /* FIXME add timeout control */
    while (CAN_GetOperationMode(CAN) != CAN_OPERATE_STOP) {}
}

static void CAN_Gpio_Init(void)
{
    #if defined(SPD1179)
    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_CAN_TXD);
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_CAN_RXD);
    #else
    PIN_SetChannel(PIN_GPIO38, PIN_GPIO38_CAN_TXD);
    PIN_SetChannel(PIN_GPIO39, PIN_GPIO39_CAN_RXD);
    #endif
}

static void CAN_Config(void)
{
    uint32_t u32Data ;
    uint32_t u32DateSegment1 ;
```

```
double f64NBps ;
double f64DBps ;

uint32_t u32Prescaler;

/* Set IO swap */
CAN_DisablePinSwap(CAN) ;

/* Set FDCAN fromat support */
CAN_EnableFDFormat(CAN);

/* Set ISO-CAN frame support */
CAN_DisableNonIsoMode(CAN);

/* Set protocol exception mode */
CAN_EnableProtocolException(CAN);

/* Set restricted mode */
CAN_DisableRestrictedMode(CAN);

/* Set monitor mode */
CAN_DisableMonitorMode(CAN);

/* Set transmission pause function */
CAN_EnableTransmissionPause(CAN);

/* Set RXD edge filter during integration status */
CAN_EnableEdgeFilter(CAN);

/* Enable re-tran message afte lost arbitration or error */
CAN_EnableAutoRetransmission(CAN);

/* Enable auto start bus-off recovery sequence after bus-off */
CAN_EnableAutoBusOn(CAN);

/* Set nominal bit baud rate
 * 1MHz for 100MHz clock
 **/
CAN_SetNominalBitRatePrescaler(CAN, 1);
CAN_SetNominalBitPhaseBufferSegment1(CAN, 79);
CAN_SetNominalBitPhaseBufferSegment2(CAN, 20); /* sample point: 80% */
CAN_SetNominalBitResyncJumpWidth(CAN, 8);

/* Set data bit baud rate */
if ( CAN_IsEnableFDFormat(CAN))
{
    u32Prescaler = CAN_GetNominalBitRatePrescaler(CAN);
    /* 5Mbps for 100MHz clock */
    CAN_SetDataBitRatePrescaler(CAN, u32Prescaler);
    CAN_SetDataBitPhaseBufferSegment1(CAN, 17 );
    CAN_SetDataBitPhaseBufferSegment2(CAN, 2 ); /* sample point: 90% */
    CAN_SetDataBitResyncJumpWidth    (CAN, 2 );
}

/* Calculate nominal phase baud rate */
u32Data = CAN_GetNominalBitPhaseBufferSegment1(CAN) + 1;
u32Data += CAN_GetNominalBitPhaseBufferSegment2(CAN);
u32Data *= CAN_GetNominalBitRatePrescaler(CAN);
f64NBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;

printf("[CAN]NBps: %f\n", f64NBps ) ;
```

```
    /* Calculate data phase baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Data = CAN_GetDataBitPhaseBufferSegment1(CAN) + 1;
        u32Data += CAN_GetDataBitPhaseBufferSegment2(CAN);
        u32Data *= CAN_GetDataBitRatePrescaler(CAN);
        f64DBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;
        printf("[CAN]DBps: %f\n", f64DBps ) ;
    }

    /* Set delay befor bus-off recovery, based on nominal bit time */
    /* Set delay 1ms */
    u32Data = (uint32_t)(1.0e6 / ( 1.0e9 / f64NBps));
    if (u32Data > 65535)
    {
        u32Data = 65535;
    }

    CAN_SetBusOffRecoveryDelay(CAN, u32Data) ;

    /* Set transmitter delay compensation for FD frame */
    CAN_EnableTransmitterDelayCompensation(CAN);

    /* Set transmitter delay compensation offset and window for FD frame */
    u32DateSegment1 = CAN_GetDataBitPhaseBufferSegment1(CAN);
    CAN_SetTransmitterDelayOffset(CAN, u32DateSegment1 + 1 );
    CAN_SetTransmitterDelayWindow(CAN, u32DateSegment1 + 1 );

    /* Set message RAM parity check */
    CAN_EnableParityCheck(CAN);

    /* Set timestamp */
    CAN_EnableTimestamp(CAN);
}


static void CAN_Printf_Message(CAN_MessageTypeDef *msg)
{
    int i;

    if (msg->eType == CAN_FRAME_DATA)
    {
        printf("    ") ;
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            printf("0x%02X,", msg->pu8Data[i]);
            if (i % 8 == 7)
            {
                printf("\n    ");
            }
        }
    }

    printf("\n");
}

static ErrorStatus CAN_CheckMessageData(CAN_MessageTypeDef *msg)
{
    int i;
    ErrorStatus status = SUCCESS;

    if (msg->eType == CAN_FRAME_DATA)
    {
```

```
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            if (msg->pu8Data[i] != i)
            {
                printf("Data Error: pu8Data[%d] != %d\n", i, i);
                status = ERROR;
            }
        }
    }

    return status;
}

uint8_t data[128];


/*****************************************************************************************
 *
 * @brief      In this case, the CANFD receive Standard frame data.
 *

*****************************************************************************************/
int main(void)
{
    ErrorStatus status;

    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("CAN: Receive Message\n");

    /* CAN Clk Enable */
    CAN_Clk_Enable();

    /* CAN Disable */
    CAN_Module_Disable();

    /* CAN Gpio Init */
    CAN_Gpio_Init();

    /* CAN config */
    CAN_Config();

    /* Init message RAM contents before enable CAN module */
    CAN_InitMessageRAM(CAN);

    /* CAN run */
    CAN_Enable(CAN);


    /* Init mailbox for receive message1 */
    message1.eDataDir    = CAN_MSG_DATA_RX;
    message1.eRmtRspEn   = DISABLE;
    message1.eIntEn      = ENABLE;
    message1.eEobEn      = DISABLE;
```

```
    message1.eOverwriteEn= ENABLE;
    message1.eMaskRtrEn  = ENABLE;
    message1.eMaskIdeEn  = ENABLE;

    message1.u32Id       = 0x22;
    message1.u32IdMask   = 0xff;
    message1.u8MBoxId    = 1;
    message1.eFormat     = CAN_FORMAT_FD_STD;
    message1.eType       = CAN_FRAME_DATA;

    /* DATA FIELD */
    message1.u8DataLen   = CANFD_DATA_LEN;

    status = CAN_SetMessage(CAN, &message1);
    if (status == ERROR)
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message1.u8MBoxId);
        return -1;
    }

    message1.pu8Data = (uint8_t *)(long)(& CAN->CANMBOX[message1.u8MBoxId].CANMBOXFDW[0]);

    CAN_EnableMailbox(CAN, message1.u8MBoxId);

    /* Wait receive CANFD Standard Data frame message1 */
    while (!CAN_GetMailboxControlInfo(CAN, message1.u8MBoxId, CAN_MBOX_SET_MSG_NEW)) {}

    CAN_GetMessage( CAN, &message1);

    #if DEBUG_INFO
    printf("CANFD Standard Frame Data:\n") ;
    CAN_Printf_Message(&message1);
    #endif

    status = CAN_CheckMessageData(&message1);
    if (status != SUCCESS)
    {
        printf("Receives CANFD Standard Data frame fail\n");
    }

    printf("Receives CANFD Standard Data frame success\n");

    while(1)
    {

    }
}
```

7.2.3 接收扩展帧数据

本示例中演示使用 CANFD 协议接收扩展格式数据帧信息，慢速阶段的传输速率为 1Mbps，采样点为 80%，快速阶段的传输速率为 5Mbps，采样点为 90%，其配置流程如下：

- 调用 CAN_DisableFDFormat()函数使能 CANFD 协议；
- 调用 CAN_DisableNonIsoMode()函数失能 NONISO 模式，使用默认的 ISO 模式；
- 调用 CAN_SetNominalBitRatePrescaler()、CAN_SetNominalBitPhaseBufferSegment1()、

CAN_SetNominalBitPhaseBufferSegment2()函数，根据 6.2 章节计算 CAN 的通信速率和采样点；

- 配置接收标准格式的数据帧数据；
  eDataDir= CAN_MSG_DATA_RX 配置消息传输的方向为接收；
  u32Id = 0x44 配置传输消息的标识符为 0x44；
  u32IdMask = 0xff 配置传输消息标识符屏蔽位为 0xff；
  u8MBoxId = 4 配置使用的邮箱的 ID 为 4；
  eFormat = CAN_FORMAT_FD_EXT 配置传输的消息为扩展 FD 格式；
  eType = CAN_FRAME_DATA 配置传输的消息为数据帧类型；
  u8DataLen = CANFD_DATA_LEN 配置传输数据的长度为 CANFD_DATA_LEN；
  pu8Data = (uint8_t*)(long)(&CAN->CANMBOX[message.u8MBoxId].CANMBOXFDW[0]) 获取传输的数据；
- 调用 CAN_GetMailboxControlInfo()函数等待指定的邮箱传输的消息是否更新；

接收扩展帧数据

```
#include <stdio.h>
#include <stdlib.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


#define DEBUG_INFO      1

#define CANFD_DATA_LEN        64

/* CANFD Extended Data frame message1 */
CAN_MessageTypeDef message1;

static void CAN_Clk_Enable(void)
{
    /* CAN Clk Enable */
    CLOCK_SetModuleDiv(CAN_MODULE, 1);
    CLOCK_EnableModule(CAN_MODULE);
}


static void CAN_Module_Disable(void)
{
    /* Disable run */
    CAN_Disable(CAN) ;

    /* FIXME add timeout control */
    while (CAN_GetOperationMode(CAN) != CAN_OPERATE_STOP) {}
}

static void CAN_Gpio_Init(void)
{
    #if defined(SPD1179)
    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_CAN_TXD);
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_CAN_RXD);
    #else
```

```
    PIN_SetChannel(PIN_GPIO38, PIN_GPIO38_CAN_TXD);
    PIN_SetChannel(PIN_GPIO39, PIN_GPIO39_CAN_RXD);
    #endif
}

static void CAN_Config(void)
{
    uint32_t u32Data ;
    uint32_t u32DateSegment1 ;

    double f64NBps ;
    double f64DBps ;

    uint32_t u32Prescaler;

    /* Set IO swap */
    CAN_DisablePinSwap(CAN) ;

    /* Set FDCAN fromat support */
    CAN_EnableFDFormat(CAN);

    /* Set ISO-CAN frame support */
    CAN_DisableNonIsoMode(CAN);

    /* Set protocol exception mode */
    CAN_EnableProtocolException(CAN);

    /* Set restricted mode */
    CAN_DisableRestrictedMode(CAN);

    /* Set monitor mode */
    CAN_DisableMonitorMode(CAN);

    /* Set transmission pause function */
    CAN_EnableTransmissionPause(CAN);

    /* Set RXD edge filter during integration status */
    CAN_EnableEdgeFilter(CAN);

    /* Enable re-tran message afte lost arbitration or error */
    CAN_EnableAutoRetransmission(CAN);

    /* Enable auto start bus-off recovery sequence after bus-off */
    CAN_EnableAutoBusOn(CAN);

    /* Set nominal bit baud rate
     * 1MHz for 100MHz clock
     **/
    CAN_SetNominalBitRatePrescaler(CAN, 1);
    CAN_SetNominalBitPhaseBufferSegment1(CAN, 79);
    CAN_SetNominalBitPhaseBufferSegment2(CAN, 20); /* sample point: 80% */
    CAN_SetNominalBitResyncJumpWidth(CAN, 8);

    /* Set data bit baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Prescaler = CAN_GetNominalBitRatePrescaler(CAN);
        /* 5Mbps for 100MHz clock */
        CAN_SetDataBitRatePrescaler(CAN, u32Prescaler);
        CAN_SetDataBitPhaseBufferSegment1(CAN, 17 );
        CAN_SetDataBitPhaseBufferSegment2(CAN, 2 ); /* sample point: 90% */
        CAN_SetDataBitResyncJumpWidth    (CAN, 2 );
    }
```

```
    /* Calculate nominal phase baud rate */
    u32Data = CAN_GetNominalBitPhaseBufferSegment1(CAN) + 1;
    u32Data += CAN_GetNominalBitPhaseBufferSegment2(CAN);
    u32Data *= CAN_GetNominalBitRatePrescaler(CAN);
    f64NBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;

    printf("[CAN]NBps: %f\n", f64NBps ) ;

    /* Calculate data phase baud rate */
    if ( CAN_IsEnableFDFormat(CAN))
    {
        u32Data = CAN_GetDataBitPhaseBufferSegment1(CAN) + 1;
        u32Data += CAN_GetDataBitPhaseBufferSegment2(CAN);
        u32Data *= CAN_GetDataBitRatePrescaler(CAN);
        f64DBps = CLOCK_GetModuleClock(CAN_MODULE) / u32Data;
        printf("[CAN]DBps: %f\n", f64DBps ) ;
    }

    /* Set delay befor bus-off recovery, based on nominal bit time */
    /* Set delay 1ms */
    u32Data = (uint32_t)(1.0e6 / ( 1.0e9 / f64NBps));
    if (u32Data > 65535)
    {
        u32Data = 65535;
    }

    CAN_SetBusOffRecoveryDelay(CAN, u32Data) ;

    /* Set transmitter delay compensation for FD frame */
    CAN_EnableTransmitterDelayCompensation(CAN);

    /* Set transmitter delay compensation offset and window for FD frame */
    u32DateSegment1 = CAN_GetDataBitPhaseBufferSegment1(CAN);
    CAN_SetTransmitterDelayOffset(CAN, u32DateSegment1 + 1 );
    CAN_SetTransmitterDelayWindow(CAN, u32DateSegment1 + 1 );

    /* Set message RAM parity check */
    CAN_EnableParityCheck(CAN);

    /* Set timestamp */
    CAN_EnableTimestamp(CAN);
}


static void CAN_Printf_Message(CAN_MessageTypeDef *msg)
{
    int i;

    if (msg->eType == CAN_FRAME_DATA)
    {
        printf("    ") ;
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            printf("0x%02X,", msg->pu8Data[i]);
            if (i % 8 == 7)
            {
                printf("\n    ");
            }
        }
    }

    printf("\n");
```

```
}

static ErrorStatus CAN_CheckMessageData(CAN_MessageTypeDef *msg)
{
    int i;
    ErrorStatus status = SUCCESS;

    if (msg->eType == CAN_FRAME_DATA)
    {
        for (i = 0 ; i < msg->u8DataLen; i++)
        {
            if (msg->pu8Data[i] != i)
            {
                printf("Data Error: pu8Data[%d] != %d\n", i, i);
                status = ERROR;
            }
        }
    }

    return status;
}



/*****************************************************************************
 *
 * @brief      In this case, the CANFD Receive Extended frame data.
 *
 *****************************************************************************/
int main(void)
{
    ErrorStatus status;

    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("CAN: Receive Message\n");

    /* CAN Clk Enable */
    CAN_Clk_Enable();

    /* CAN Disable */
    CAN_Module_Disable();

    /* CAN Gpio Init */
    CAN_Gpio_Init();

    /* CAN config */
    CAN_Config();

    /* Init message RAM contents before enable CAN module */
    CAN_InitMessageRAM(CAN);

    /* CAN run */
    CAN_Enable(CAN);
```

```
    /* Init mailbox for receive message1 */
    message1.eDataDir    = CAN_MSG_DATA_RX;
    message1.eRmtRspEn   = DISABLE;
    message1.eIntEn      = ENABLE;
    message1.eEobEn      = DISABLE;
    message1.eOverwriteEn= ENABLE;
    message1.eMaskRtrEn  = ENABLE;
    message1.eMaskIdeEn  = ENABLE;

    message1.u32Id       = 0x44;
    message1.u32IdMask   = 0xff;
    message1.u8MBoxId    = 4;
    message1.eFormat     = CAN_FORMAT_FD_EXT;
    message1.eType       = CAN_FRAME_DATA;

    /* DATA FIELD */
    message1.u8DataLen   = CANFD_DATA_LEN;

    status = CAN_SetMessage(CAN, &message1);
    if (status == ERROR)
    {
        printf("[CAN][u8MBoxId:%2d] Set MBOX failed", message1.u8MBoxId);
        return -1;
    }

    message1.pu8Data = (uint8_t *)(long)(& CAN-
>CANMBOX[message1.u8MBoxId].CANMBOXFDW[0]);

    CAN_EnableMailbox(CAN, message1.u8MBoxId);

    /* Wait receive CANFD Extended Data frame message1 */
    while (!CAN_GetMailboxControlInfo(CAN, message1.u8MBoxId,
CAN_MBOX_SET_MSG_NEW)) {}

    CAN_GetMessage( CAN, &message1);

    #if DEBUG_INFO
    printf("CANFD Extended Frame Data:\n") ;
    CAN_Printf_Message(&message1);
    #endif

    status = CAN_CheckMessageData(&message1);
    if (status != SUCCESS)
    {
        printf("Receives CANFD Extended Data frame fail\n");
    }

    printf("Receives CANFD Extended Data frame success\n");

    while(1)
    {

    }
}
```