华大半导体 HUADA SEMICONDUCTOR

32 位微控制器

HC32F4A0 系列的控制器局域网 CAN

适用对象

F 系列	**HC32F4A0**

目 录

1 摘要

本篇应用笔记主要介绍 HC32F4A0 系列的控制器局域网 CAN（以下称 CAN 控制器）的功能特性，还介绍了用该 CAN 控制器实现 CAN 2.0、CAN-FD（Flexible Data-Rate）和 TTCAN（Time-triggered）通信的方法。

2 CAN 控制器概述

2.1 简介

CAN 是 Controller Area Network 的缩写（以下称为 CAN），是 ISO 国际标准化的串行通信协议。HC32F4A0 系列 MCU 搭载两个 CAN 控制器，CAN1 和 CAN2，它们具有如下特性：

1) 完全支持 CAN2.0A/CAN2.0B/CAN FD 协议；
2) CAN2.0 支持最高通信波特率 1Mbit/s；
3) 支持 1~1/256 的波特率预分频，灵活配置波特率；
4) 8 个接收缓冲器：FIFO 方式；错误或者不被接收的数据不会覆盖存储的消息；
5) 1 个高优先主发送缓冲器 PTB（Primary Transmit Buffer）；
6) 3 个副发送缓冲器 STB（Secondary Transmit Buffer）：FIFO 方式；优先级仲裁方式；
7) 16 组独立的筛选器：支持 11 位标准 ID 和 29 位扩展 ID；可编程 ID CODE 位以及 MASK 位；
8) PTB/STB 均支持支持单次发送模式；
9) 支持静默模式；
10) 支持回环模式；
11) 支持捕捉传输的错误种类以及定位仲裁失败位置；
12) 可编程的错误警告值；
13) 支持 ISO11898-4 规定时间触发 CAN 以及接收时间戳。

2.2 系统框图

CAN 控制器系统框图如图 2-1 所示。

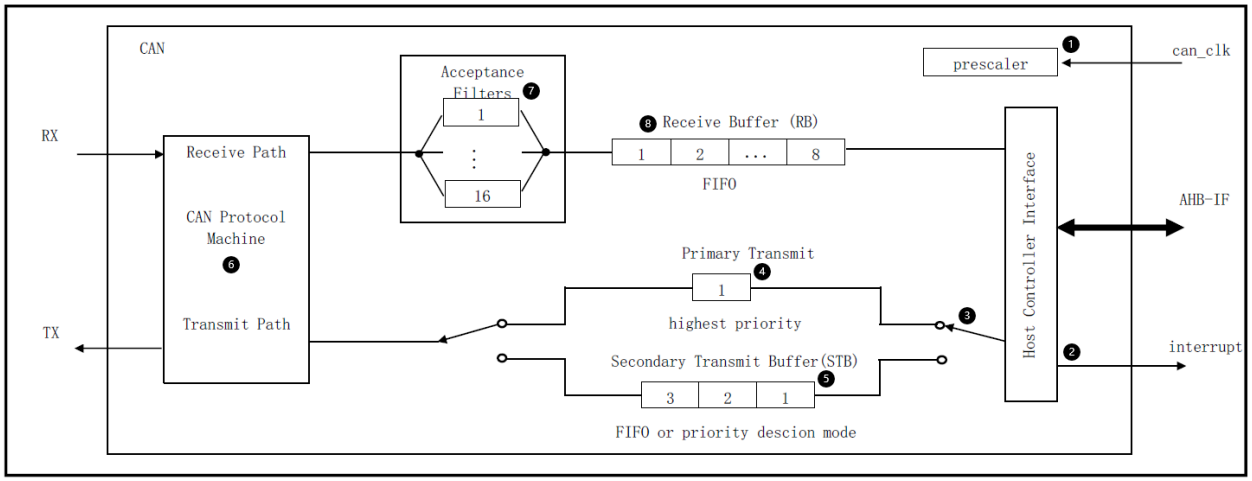

图 2-1 CAN 系统框图

说明：

1) CAN 时钟设置，通信波特率设置；

2) CAN 产生的各种中断；

3) 发送缓冲器选择；

4) 1 个主发送缓冲器 PTB（具有最高优先级）；

5) 3 个副发送缓冲器 STB；

6) CAN 收发控制器；

7) 16 个接收筛选器。每个筛选器可分别配置不同的 ID，ID Mask 以及 IDE（IDentifier Extension bit）；

8) 8 接收缓冲器。CAN 收发控制器收到的数据通过接收筛选器后，才能保存在接收缓冲器中。

2.3 管脚

CAN 控制器的管脚配置如表 2-1 所示（表中 n=1，2）。

管脚名称	方向	功能描述
CANn_RX	输入	CAN 接收数据信号
CANn_TX	输出	CAN 发送数据信号
CANn_TST_SAMPLE	输出	仅用于观测采样位置（采样点后一周期的通信时钟宽度）
CANn_TST_CLOCK	输出	仅用于观测波特率（一位开始前一周期的通信时钟宽度）

表 2-1 CAN 控制器的管脚

3 CAN 控制器的功能及应用

3.1 操作模式

HC32F4A0 系列 MCU 的 CAN 控制器有两种操作模式，复位模式（CFG_STAT.RESET=1）和动作模式（CFG_STAT.RESET=0）。在初始化时，应首先在复位模式下配置只能在复位模式下操作的寄存器（详见用户手册关于寄存器的描述），然后退出复位模式，在动作模式下配置其余寄存器。

3.2 工作模式

3.2.1 回环模式

CAN 控制器支持两种回环模式：内部回环和外部回环。两种回环模式都可以接收自己发出的数据帧，一般用于测试用途。

内部回环模式，模块内部将接收数据线连接到发送数据线，并且发送数据不输出。内部回环模式下，节点会生成自应答信号以避免 ACK 错误。

外部回环模式保持和收发器的连接因此发送的数据仍能出现在 CAN 总线上，在收发器的帮助下，CAN 控制器能收到自己发送的数据。外部回环模式可以通过寄存器位 RCTRL.SACK 位来决定是否生成自应答信号，RCTRL.SACK 为 0 时，不生成自应答信号，为 1 时，生成自应答信号。

外部回环模式，在 RCTRL.SACK 为 0 时，会出现以下两种情况：

1) 其它节点也收到本节点发送的数据帧并发送应答信号，该情况下本节点能够成功收发数据；

2) 如果没有其它节点返回应答信号，则会产生应答错误，会重新发送数据并增加错误计数器。此时推荐采用单次发送模式。

从回环模式返回到正常模式时，除了清除模式位以外，还需要软件复位 CAN 控制器。

3.2.2 静默模式

静默模式可以用来监控 CAN 网络数据。在静默模式下，可以从 CAN 总线接收数据，不向总线发送任何数据。将寄存器位 TCMD.LOM 置 1，使 CAN 总线控制器进入静默模式，将其清 0 可以离开静默模式。

外部回环模式可以和静默模式组合成外部回环静默模式，此时 CAN 控制器可以认为是一个安静的接收者，但在有必要的时候可以发送数据。外部回环静默模式下，帧包含自应答信号允许被发送，但是该节点不会产生错误标志和过载帧。

3.2.3 模式设置

驱动程序将 CAN 控制器的工作模式分为以下五种：

- 普通模式
- 静默模式
- 内部回环模式
- 外部回环模式
- 外部回环静默模式

通过 CAN 控制器的初始化结构体参数 u8WorkMode 来指定工作模式，或通过 API CAN_SetWorkMode()进行指定。

3.3 时钟及波特率

3.3.1 时钟配置

CAN 控制器有两个时钟，控制逻辑时钟和通信时钟。控制逻辑时钟用 PCLK1，最高 120MHz；通信时钟可选如下时钟源，最高 80MHz，只用于产生波特率。

1) 系统时钟的 2/3/4/5/6/7/8 分频
2) PLLHQ/R
3) PLLAP/Q/R
4) XTAL

在应用时，通过调用时钟模块的 API 来设置时钟频率，用 CLK_ClkDiv()来设置控制逻辑时钟；用 CLK_CAN_ClkConfig()来指定通信时钟，当通信时钟选择 PLLHQ/R、PLLAP/Q/R 或 XTAL 时，必须配置并使能这些时钟源。

3.3.2 波特率设定

3.3.2.1 CAN 协议的位时序

在设定波特率之前，先简单介绍一下 CAN 协议的位时序。

CAN 总线上的一个位可分为 4 段：

- 同步段（Synchronization Segment，以下简称 SS）
- 传播时间段（Propagation Time Segment，以下简称 PTS）
- 相位缓冲段 1（Phase Buffer Segment 1，以下简称 PBS1）
- 相位缓冲段 2（Phase Buffer Segment 2，以下简称 PBS2）

这些段又由可称为 Time Quantum（以下简称 TQ）的最小时间单位构成。关于各段的详细描述，请参考 ISO11898-1。

1 位分为 4 个段，每个段又由若干个 TQ 构成，这称为位时序，如图 3-1 所示。

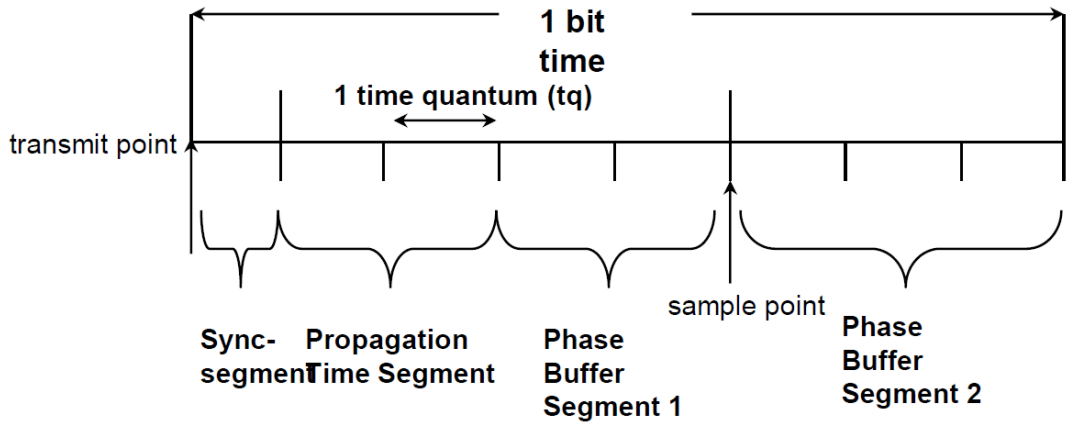

图 3-1 CAN 位时序（8TQ）

于是，可通过下面的公式计算波特率：

$$Baudrate = CANClock \div ClockPrescaler \div (SS + PTS + PBS1 + PBS2)$$

上述公式，Baudrate 是通信波特率（bps），CANClock 是 CAN 通信时钟频率（Hz），ClockPrescaler 是 CAN 通信时钟分频数。如 CAN 通信时钟为 8MHz，时钟分频数为 2，以图 3-1 所示的时序计算波特率如下：

$$Baudrate = 8000000 \div 2 \div (1 + 2 + 2 + 3) = 500000bps$$

3.3.2.2 CAN 控制器的波特率设定

CAN 控制器，将段 SS、PTS 和 PBS1 统称为 segment1，将 PBS2 称为 segment2。如图 3-2 所示。

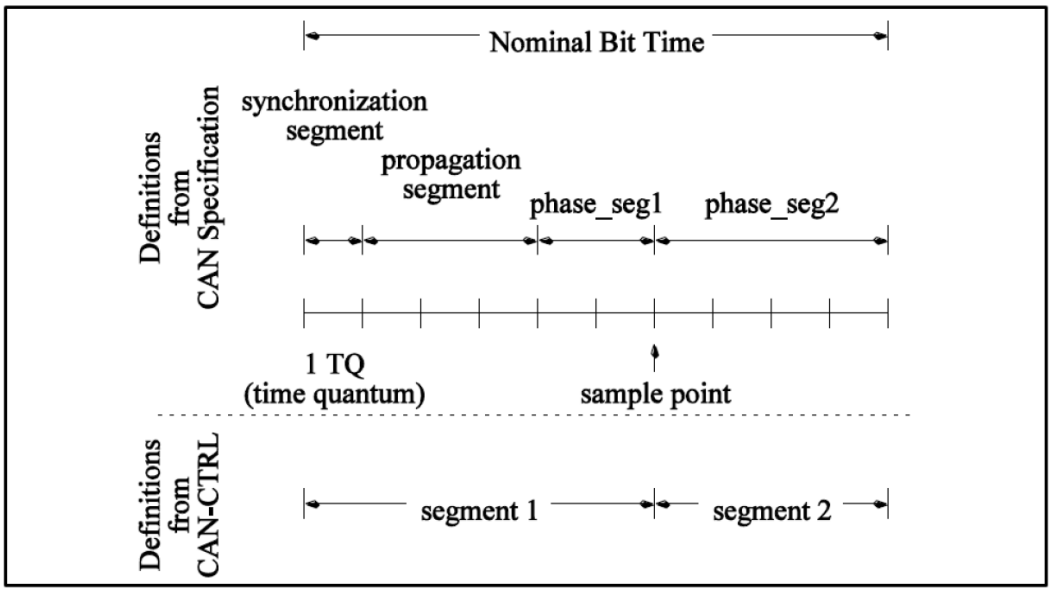

图 3-2 CAN 控制器的位时序

图 3-2 的虚线上部分为 CAN 协议规定的位时序，虚线下部分为 CAN 控制器定义的位时序。其中 segment1 和 segment2，在用 CAN2.0 通信时通过寄存器 SBT 设定，在用 CAN-FD 通信时通过寄存器 FBT 设定。

由 CAN 协议的波特率计算公式，可得 CAN 控制器的波特计算公式如下：

$$Baudrate = CANClock \div ClockPrescaler \div (segment1 + segment2)$$

采样点：

$$SamplePoint = segment1 \div (segment1 + segment2)$$

TQ 时间：

$$TQTime = 1000000(us) \div Baudrate(bps) \div (segment1 + segment2)$$

CAN 控制器的驱动程序提供了一个结构体来设置位时序，如程序清单 3-1 所示。

```
typedef struct
{
   uint32_t u32SEG1;
   uint32_t u32SEG2;
   uint32_t u32SJW;
   uint32_t u32Prescaler;
} stc_can_bt_cfg_t;
```

程序清单 3-1 CAN 控制器位时序配置结构体

其中，u32SEG1、u32SEG1 和 u32Prescaler 分别对应波特率计算公式中的 segment1、segment2 和 ClockPrescaler。在应用时，应严格按照 ISO11898-1 所描述的位时序要求来设置 segment1、segment2 和 SJW，如表 3-1 所示。

Parameter	Not FD enabled	FD enabled		
		Separate prescalers	Shared prescaler	Separate or shared
	Nominal bit time	Nominal bit time	Nominal bit time	Data bit time
Prescaler m	1 to 32	1 to 32	1 to 32	
Sync_Seg	1 time quantum(N)	1 time quantum(N)	1 time quantum(N)	1 time quantum(D)
Prop_Seg	1 to 8 time quanta(N)	1 to 48 time quanta(N)	1 to 96 time quanta(N)	0 to 8 time quanta(D)
Phase_Seg1	1 to 8 time quanta(N)	1 to 16 time quanta(N)	1 to 32 time quanta(N)	1 to 8 time quanta(D)
Phase_Seg2	2 to 8 time quanta(N)	2 to 16 time quanta(N)	2 to 32 time quanta(N)	2 to 8 time quanta(D)
SJW	1 to 4 time quanta(N)	1 to 16 time quanta(N)	1 to 32 time quanta(N)	1 to 8 time quanta(D)

表 3-1 ISO11898-1 CAN 位时序之各段 TQ 的范围

在使用 CAN-FD 通信时，建议通信时钟选取 20MHz、40MHz 或 80MHz，并参考用户手册中的波特率设定建议表进行波特率设置。

CAN 控制器的驱动程序 API CAN_StructInit()，提供了 CAN2.0 和 CAN-FD 的波特率配置的示例。其中，CAN2.0 的波特率为 500Kbps，基于 40MHz 的 CAN 通信时钟，采样点为 80%；CAN-FD 波特率为 2Mbps，基于 40MHz 的 CAN 通信时钟，第一采样点和第二采样点均为 80%。

3.4 发送缓冲器

CAN 控制器提供两种发送缓冲器用于发送数据，主发送数据缓冲器 PTB 和副发送缓冲器 STB。

PTB 具有最高的优先级，其发送能推迟 STB 发送，但是已经赢得仲裁并开始发送的 STB 不能够被 PTB 发送推迟。PTB 只能缓冲一帧数据。

STB 优先级比 PTB 低，但可以缓冲 3 帧数据，且 STB 内 3 帧数据可以工作在 FIFO 模式或者优先级仲裁模式。STB 中的 3 帧说数据可以通过 TCMD 寄存器的 TSALL 位设定为 1 全部发送，在 FIFO 模式下，最先写入的数据先发送，在优先级模式下，ID 小的数据先发送。

PTB 和 STB 可以通过 TBUF 寄存器进行访问。通过 TCMD 寄存器的 TBSEL 位选择 PTB 或者 STB，TBSEL=0，选择 PTB，TBSEL=1，选择 STB。通过 TCTRL 寄存器的 TSNEXT 位选择 STB 中的下一个 SLOT。对应关系如图 3-3 所示。

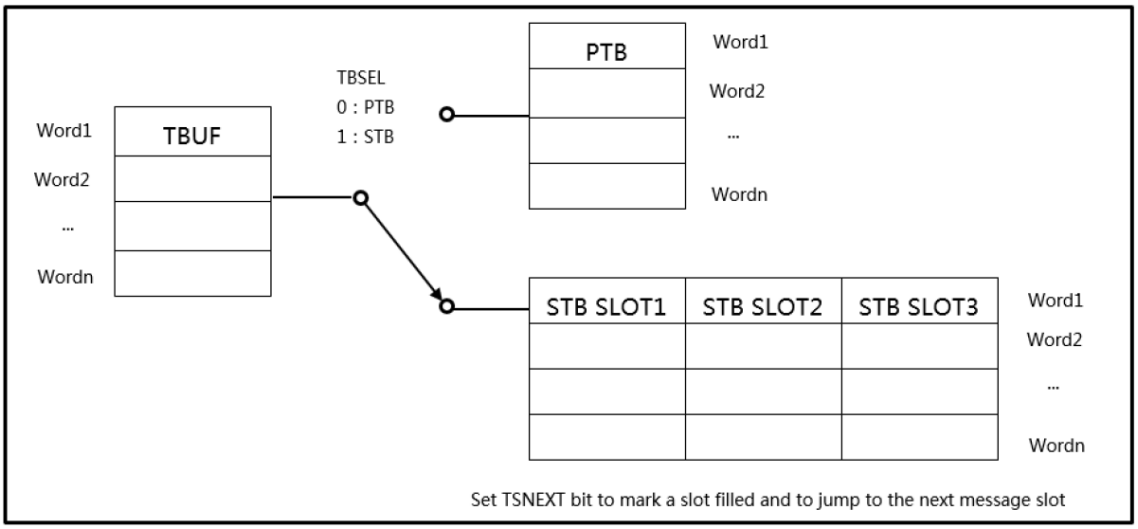

图 3-3 通过寄存器 TBUF 写发送缓冲器

3.5 接收缓冲器

CAN 控制器提供 8 个接收缓冲器用于存储接收到的数据，工作在 FIFO 模式。

接收缓冲器通过 RBUF 寄存器来读取接收到的数据（图 3-4），总是最先读取最早接收到的数据；通过 RCTRL 寄存器的 RREL 设置为 1 释放已经读取的接收缓冲器，并指向下一个接收缓冲器。

图 3-4 通过寄存器 RBUF 读取接收缓冲器

接收缓冲器有 4 中状态：

- 空
- 非空但小于 LIMIT.AFWL 编程值
- 大于等于 LIMIT.AFWL 编程值但未满
- 满

这 4 种状态，可通过读取寄存器 RCTRL.RSTAT 获取。CAN 控制器的驱动程序提供了获取接收缓冲器状态的 API CAN_GetRBFullStatus()。

在有些应用场景中，我们可能需要接收缓冲器接收到的帧的个数达到指定数值时，产生一个中断。CAN 控制器的寄存器 LIMIT.AFWL 即可设定这样一个数值。如设置 LIMIT.AFWL 为 n

（n=1~8）时，当接收缓冲器接收到的帧的个数大于等于 n 时，标志位 RTIF.RAFIF（接收将满标志位）置位，接收将满中断产生，应用程序可在中断处读取接收到的数据。

3.6 接收筛选器

CAN 控制器提供 16 组 32 位筛选器用于过滤接收到的数据从而降低 CPU 负荷。

筛选器可以支持标准格式 11 位 ID 或者扩展格式 29 位 ID。接收到的数据只要通过 16 组筛选器的任意一组，则被接收。接收到的数据存储在接收缓冲器中，否则数据不被接收，也不被存储。

每组筛选器有一个 32 位 ID CODE 寄存器和一个 32 位 ID MASK 寄存器，ID CODE 寄存器用于比较接收到 ID，而 ID MASK 寄存器用于选择比较的 ID 位。对应的 ID MASK 位为 1 时，不比较该位的 ID CODE。如 ID 为 b’11000011，ID MASK 为 b’00000011，那么筛选器将不会比较 ID 的低两位，即，可接收所有 ID 为 b’110000xx（b’xx 为任意值）的帧。

CAN 控制器的驱动程序提供了筛选器配置结构体（如程序清单 3-2 所示），并通过调用 API CAN_AFConfig()来配置筛选器。

```
typedef struct
{
   uint32_t u32ID;
   uint32_t u32IDMsk;
   uint32_t u32MskType;
} stc_can_af_cfg_t;
```

程序清单 3-2 CAN 控制器的筛选器配置结构体

在筛选器配置结构体中：

- u32ID：ID，11 位标准 ID 或 29 位扩展 ID
- u32IDMsk：ID 掩码，筛选器不比较 ID 中对应 u32IDMsk 位为 1 位
- u32MskType：筛选器可接收的 ID 类型：标准 ID 或扩展 ID 或两者

3.7 数据发送

CAN 控制器在发送数据时有两种发送模式，自动重发模式和单次发送模式。发送模式可在初始化时配置，也可调用驱动程序的 API CAN_SetTransMode()在需要时修改。

在开始发送前必须保证 PTB 或者 STB 中至少有一帧数据已被装载，PTB 发送过程中 TPE 被锁定，STB 的填充情况可以通过 TSSTAT 位确认。发送数据设定步骤如下：

1. 设定 TBSEL 从 PTB 和 STB 中选择发送缓冲器；

2. 通过 TBUF 寄存器写需要发送的数据；
3. 如果选择的时 STB，设置 TSNEXT=1 以完成 STB SLOT 的装载；
4. 发送使能：
   - PTB 发送使用 TPE；
   - STB 发送使用 TSALL 或者 TSONE。
5. 发送完成状态确认：
   - PTB 发送完成使用 TPIF，TPIE 用于使能 TPIF；
   - STB 采用 TSONE 发送完成时使用 TSIF，TSIE 用于使能 TSIF；
   - STB 采用 TSALL 发送完成时使用 TSIF，此时需要设定的全部 STB SLOT 数据发送完成后，TSIF 才置位，TSIE 用于使能 TSIF。

驱动程序的 API CAN_TransData()可用来发送 CAN2.0（TTC 未使能时）和 CAN-FD 的数据。该 API 可指定发送用的发送缓冲器（PTB 或 STB），每次发送一帧数据。

当一次写入多个数据帧到 STB 且使能全部发送时，可通过寄存器位 TCTRL.TSSTAT 获取当前 STB 的状态。成功发送一帧数据，TCTRL.TSSTAT 的值自减 1，可结合 STB 的优先级模式，知道当前发送成功的数据帧或发送失败的数据帧。

3.8 取消数据发送

可以通过 TPA 或者 TSA 取消已请求但还没有被执行的数据发送。取消数据发送会出现以下几种情况：

1. 仲裁中：
   - 节点仲裁失败，则取消数据发送；
   - 节点仲裁成功，则继续发送。
2. 数据发送中：
   - 成功发送数据且收到 ACK，对应的标志和状态正常置位。数据发送不取消；
   - 成功发送数据但没有收到 ACK，数据发送取消，错误计数器增加；
   - TSALL=1 设定的发送数据，正在发送的 STB SLOT 数据正常发送，没有开始发送的 STB SLOT 被取消。

取消数据发送的结果有以下两种情况。

1. TPA 释放 PTB，且使 TPE=0；
2. TSA 释放一个 STB SLOT 或者全部 STB SLOT 取决是 TSONE 还是 TSALL 使能的发送。

3.9 数据接收

接收筛选器组可以过滤掉不需要的接收数据，减少中断的发生和 RB 的读取，从而降低 CPU 负荷。接收数据设定步骤如下：

1. 设定筛选器组；
2. 设定 RFIE，RAFIE 和 AFWL；
3. 等待 RFIF 或者 RAFIF；
4. 通过 RBUF 从 RB FIFO 中读取最早接收到的数据；
5. 设置 RREL=1，选择下一个 RB SLOT；
6. 重复 4，5 直到通过 RSTAT 确认 RB 为空。

一般地，步骤 1 和 2 在初始化 CAN 控制器时设定；可以以轮询的方式等待标志位 RFIF 或 RAFIF 置位，并读取数据，也可以在接收中断或接收中断将满中断中读取数据。CAN 控制器的驱动程序提供的 API CAN_ReceiveData()即用于读取接收到的数据，可一次读取多帧数据。

如果接收器满之后再收到数据，就会产生接收缓冲器溢出（Receive buffer Overflow）根据 RCTRL.ROM 的设置，这个新数据，可能覆盖旧数据（被保存），也可能被丢弃。如果这个新数据被保存，就产生 overrun（RTIF.ROIF 置位），否则不产生 overrun。

3.10 错误处理

CAN 控制器一方面可以自动处理部分错误，比如自动重发数据或者丢弃接收到含有错误的帧，另一方面通过中断将错误向 CPU 报告。

CAN 节点有以下三种错误状态：

- 主动错误（Error-active）：节点检测到错误时自动发送主动错误标志
- 被动错误（Error-passive）：节点检测到错误时自动发送被动错误标志
- 节点关闭：关闭状态下此节点不再影响整个 CAN 网络

CAN 控制器提供 TECNT 和 RECNT 两个计数器用于计数错误。TECNT 和 RECNT 计数器按照 CAN 协议规定的规则进行增减：接收错误产生时 RECNT 增加，正确接收到数据帧 RECNT 减少；发送错误产生时 TEC 增加，正确发送一帧数据帧 TEC 减少。另外，控制器还提供可编程的 CAN 错误警告 LIMIT 寄存器用于产生错误中断通知 CPU。

错误帧在总线信号传输中起着信号灯的作用，接收和发送过程中如果检测到通信出错，便会发送错误帧，错误帧由错误标识符以及错误界定符构成。其中错误标识符分为两种：“主动错误”和“被动错误”。

主动错误状态下，只要检查到错误，它立即“主动地”发出错标识。所谓“出错标识”，它本身就是一个“错误的位序列”（连续的 6 个显性位，不满足 CAN 协议的“最多 5 个连续的同性位”要求），目的是“主动地”告诉大家：节点检查到错误后，主动发出错误标识，以通知其他节点它检查到了错误。

如果处于被动状态下，检查到错误，它只能等待等主动错误节点发出错误标识，等待的时候它不能通过总线发送数据，直到识别出由主动错误节点发出的“错误的位序列”，才可以发送数据，去竞争总线。

为了避免某个设备因为自身原因（例如硬件损坏）导致无法正常收发数据而不断地破坏数据帧，从而影响其他正常节点通讯，CAN 总线规范中规定每个 CAN 控制器都有一个发送错误计数器和一个接收计数器。根据计数值不同 CAN 节点会处于不同的设备状态，状态之间的转换关系如图 3-5。

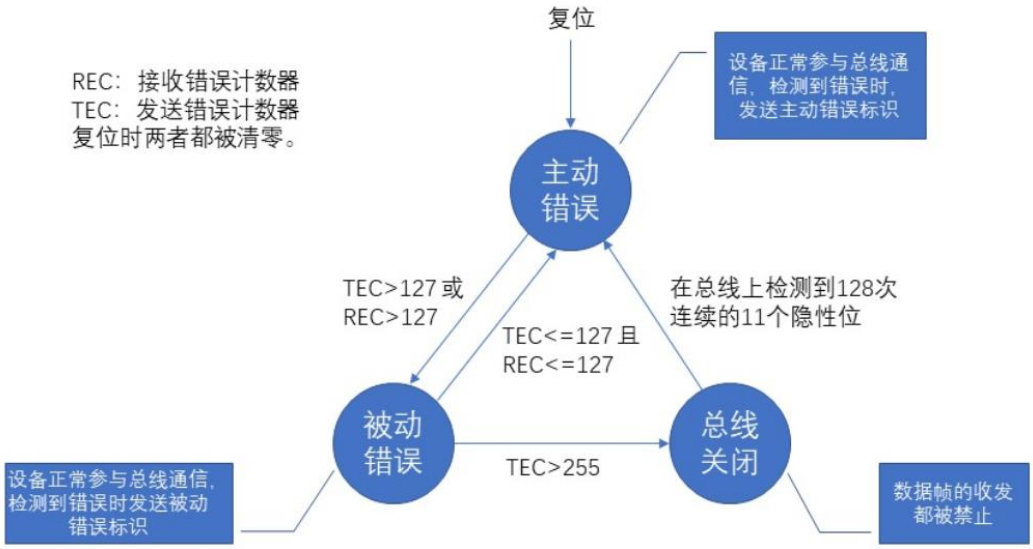

图 3-5 CAN 总线错误状态转换关系图

CAN 通信过程中有以下 5 种错误类型，CAN 控制器的寄存器 EALCAP.KOER 位用于识别这些错误类型。

- 位错误（Bit error）：发送节点在发送数据的同时也会对发出的比特流采样回收，如果监测到总线电平与正在发送的电平不符，将会发生位错误；
- 形式错误（Form error）：当传输的数据帧格式与任何一种合法的帧格式不符时会发生该类形式错误；
- 填充错误（Stuff error）：为解决异步串行通讯中共有的缺点——各节点通讯时钟不同步，CAN 总线采用了一种同步的方式，规定每隔 5 个位的时间长度进行一次同步，当传输信号连续 5 个位是相同的，就要插入一个点评相反的位（称为“填充位”），如果电缆

上传输的信号不满足这一规则，则会出现填充错误；

- 应答错误（ACK error）：如果在 ACK 段发送节点没有接收到接收节点发出的应答（显性位），将发生应答错误（Acknowledgment Error），所以当单个节点发送数据帧时会发生该类错误；
- CRC 错误（CRC error）：发送端发送的 CRC 值由发送器计算得出，接收器采用同样的 CRC 算法，计算出接收端 CRC 值，若接收器计算的 CRC 值与接收到的 CRC 值不符，将会产生该类错误。

3.11 节点关闭

当发送错误数大于 255 时，CAN 节点自动进入节点关闭（Bus-off）状态从而不参与 CAN 通信（如图 3-5 所示），直到返回到错误主动状态。可以通过寄存器 CFG_STAT.BUSOFF 位确认 CAN 节点关闭状态。BUSOFF 被置位的同时 EIF 中断产生。

当 CAN 节点进入节点关闭状态（Bus-off）后，该节点不允许对总线产生任何影响，此时节点既不会发送任何报文或是发送 ACK 应答、错误帧、过载帧。节点是否能够接收总线报文，取决于实际的应用情况。

CAN 从节点关闭状态恢复到错误主动状态有以下两种方法：

- 上电复位;
- 接收到连续 128 个 11 位的隐形位序列（恢复序列）。

节点关闭状态下，寄存器 TECNT 值保持不变，RECNT 用于计数恢复序列。从节点关闭状态恢复后，TECNT 和 RECNT 被复位为 0。

注意，软复位不能使节点从节点关闭状态恢复到主动错误状态。

3.12 CAN-FD

CAN-FD（CAN with Flexible Data-Rate，可变的数据率）是对 ISO11898-1 规定的原有 CAN 协议的扩展，用于响应汽车网络中日益增加的对带宽的需求。CAN-FD 继承了 CAN 总线的主要特性，提高了 CAN 总线的网络通信带宽，改善了错误帧漏检率，同时可以保持网络系统大部分软硬件特别是物理层不变。

3.12.1CAN-FD 的波特率

CAN-FD 采用两种位速率，从控制场中的 BRS（Bit Rate Switch）位到 ACK 场之前（含 CRC 界定符）为可变位速率，其余部分为传统 CAN 总线（CAN2.0A/B）用的位速率，两种速率各有一套位时间定义寄存器，它们除了采用不同的位时间单位 TQ 外，位时间各段的分配比例也

可不同。CAN 控制器的寄存器 SBT 用于设置传统 CAN 总线的位速率，寄存器 FBT 用于设置 CAN-FD 的可变位速率。设置 CAN-FD 的可变位速率时，可参考用户手册波特率设定章节的“20MHz/40MHz/80MHz 通信时钟时波特率设定建议”。

3.12.2CAN-FD 的 TDC 和 RDC

CANFD 通信时，数据的延迟可能超过一位时间，因此需要补偿。TDC（Transmitter Delay Compensation）和 RDC(Receiver Delay Compensation)用于数据延迟的补偿。其中 TDC 需要软件控制开关，而 RDC 不需要，自动有效。TDC 中采用辅助采样点 SSP（Secondary Sample Point）的方式去补偿数据延迟，如图 3-6 所示。

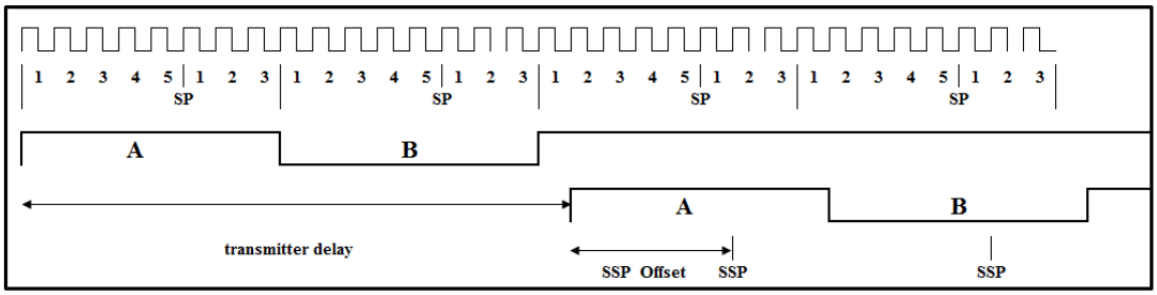

图 3-6 TDC 功能示意

软件使能 TDC 功能时，本控制器可以自动确定 transmitter delay，通过设定寄存器 TDC.SSPOFF 来设定 SSP Offset。建议 SSP Offset 设定值和$tF_segment1$ 设定值相同。

3.12.3CAN-FD 的模式

CAN 控制器支持两种 CAN-FD 模式，“Bosch CAN-FD”模式和“11898-1 2015 CAN-FD”模式，通过寄存器 TCTRL.FD_ISO 进行设置。只有 CAN-FD 模式相同的两个 CAN-FD 节点才可以相互收发 CAN-FD 帧。

CAN 控制器的 CAN-FD 功能可禁止或使能。CAN-FD 功能被禁止时，即使在包含 CAN-FD 节点的网络中收到 CAN-FD 帧，接收器也会自动屏蔽这些帧，不返回 ACK。等到总线空闲时，再发送或接收下一帧 CAN2.0 的帧。

3.13 TTCAN

3.13.1TTCAN 介绍

TTCAN（Time-triggered CAN）协议是一种在 ISO 11898-1 规定的 CAN 数据链路层顶部之上的高层协议，可基于标准 CAN 物理层协议实现。ISO11898-4 标准详细介绍了 TTCAN 协议在 CAN 协议的基础上所增加的时间触发功能部分。TTCAN 协议的时间触发功能的含义包括两层：Level1 要求网络各节点建立一个各自的本地时钟（Local Time），并通过时间主节点

（Time Master）发送特定 ID 的参考消息（Reference Message）实现各节点的时钟同步，即网络各节点每收到一次参考消息，各节点本地时钟被同步一次；在时钟同步的基础上，各节点按照系统矩阵（System Matrix，如图 3-7）的调度安排，在规定的时间窗（Time Window）内完成相应的任务，从而实现周期性地传输消息的。Level2 在本地时钟基础上产生一个系统全局时钟（Global Time）。

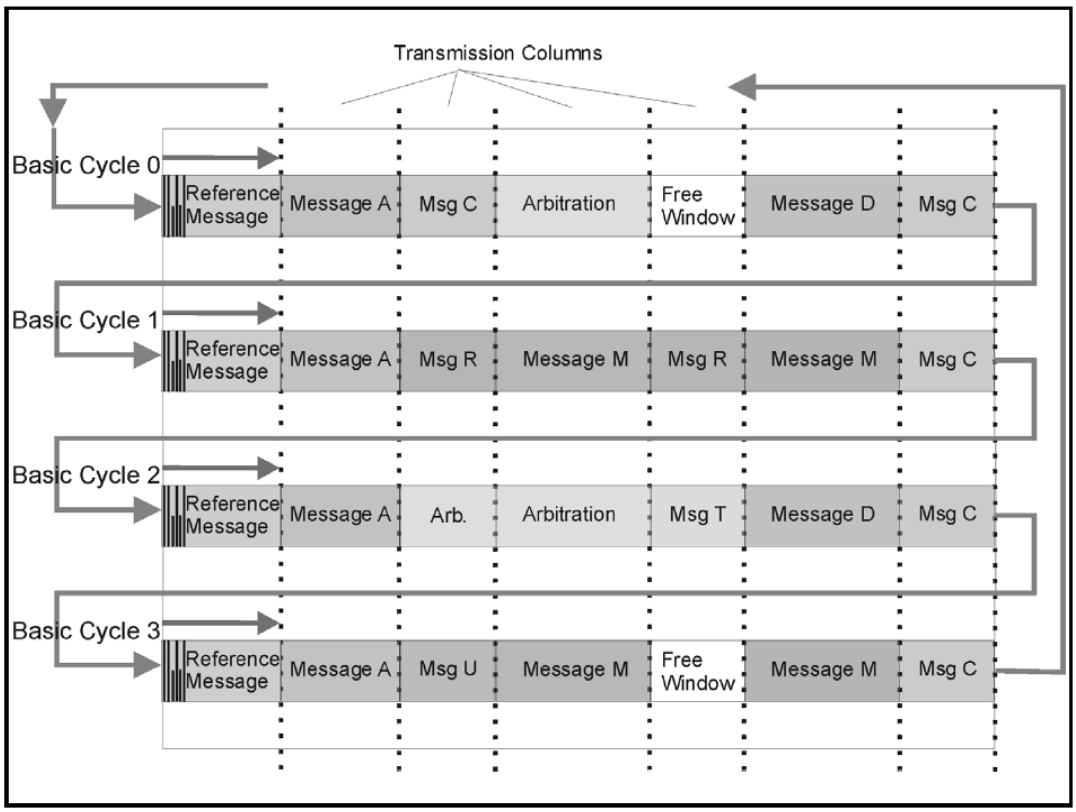

图 3-7 TTCAN 系统矩阵

为了避免各节点周期型消息之间的冲突，TTCAN 协议下的网络在通讯前就需要根据整个系统所有节点的消息制定出合理的系统矩阵，通过系统矩阵对总线的通讯时间进行划分。系统矩阵又被称作静态调度表，在整个通讯过程中各节点在时钟同步的条件下按照系统矩阵的调度安排来获得对总线的使用权，在规定的时间窗中完成相应的任务，从而保证在总线的通讯过程中周期型消息的实时性。整个 TTCAN 总线网络的通讯实际上是各节点在系统矩阵的调度安排下，周期性地完成系统矩阵中所规定的任务，即系统矩阵的周期性循环。

系统矩阵（Matrix Cycle）由多个基本周期（Basic Cycle）组成，一个基本周期对应系统矩阵的一行，系统矩阵的整个长度被称为一个矩阵周期。根据 ISO11898-4 标准，TTCAN 协议下的系统矩阵应当由 2 的 n 次方（n=0、1、2、3、…）个基本周期组成，当 n=0 时，该系统矩阵即为单行矩阵。系统矩阵中每个基本周期由参考消息开始，到下一条参考消息结束，并且每个基本周期中的参考消息需要反应出该基本周期在矩阵中所处的行数。一个基本周期的长度实际上

就是参考消息的周期长度；一个矩阵周期的长度可以是所有消息发送周期的最小公倍数，或者最小公倍数的整数倍。

在 TTCAN 网络中，从消息的实时性要求来分，总线上所传输的消息分为两种类型：周期型消息和非周期型消息。周期型消息是指具有固定的发送周期，对实时性的要求较高，对系统而言比较重要的消息；周期型消息采用时间触发的方式进行发送，并且在规定的时间段内只允许发送一条该消息，不与总线上的其他消息因为争夺总线的使用权而发生冲突；非周期型消息指的是对实时性要求不高，没有规定发送周期的消息；与 CAN 总线的仲裁机制相同，TTCAN 网络中的非周期型消息在规定的时间段内需要与其它的非周期型消息互相竞争，争夺总线的使用权。

TTCAN 协议中，系统矩阵的一行称作一个基本周期，而每个基本周期又由多个时间窗组成，并且处于同一列的时间窗构成了系统矩阵的列。每个基本周期的时间窗组成可能不同，各时间窗的长度及所完成的任务也可能不同，但同一列的各个时间窗的长度应当相同。时间窗共分三种类型：独占窗、仲裁窗和自由窗。独占窗主要用于周期型消息的发送，并且只允许一个节点发送一条周期型消息；仲裁窗对应的时间段内允许多个节点发送多条非周期型消息，通过 CAN 总线的仲裁机制决定最终由一个节点发送消息；自由窗作为系统扩展时使用。在独占窗中，不允许发送失败的消息重发；在仲裁窗中，也不允许发送失败的消息重发，两个仲裁窗可以合并的情况除外，也就是说，两个相邻的仲裁窗可以合并为一个仲裁窗（如图 3-8），在合并的仲裁窗内允许重发失败的消息。

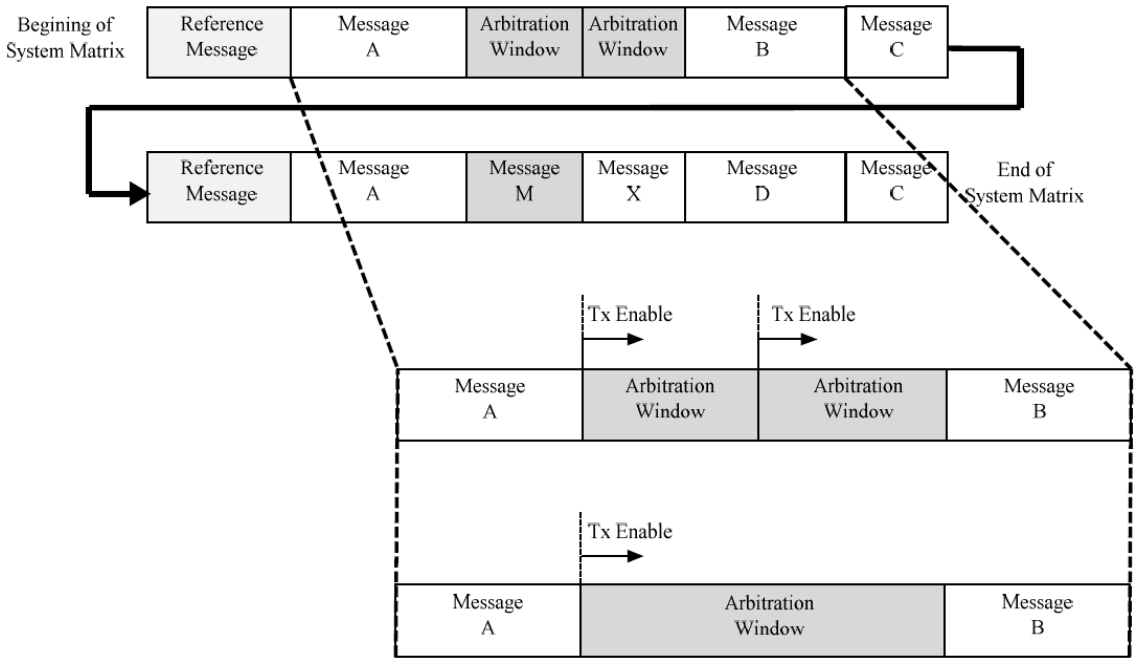

图 3-8 TTCAN 合并仲裁窗

对于一个采用 TTCAN 协议通讯的总线网络，其通讯过程实际上是系统矩阵的不断循环，因此，制定出合理的系统矩阵对网络的实时性和总线带宽利用率等性能有着决定性的影响。用户需基于 ISO11898-4 标准和实际应用场景，制定出合适的系统矩阵，以及合理设定各个时间窗的长度，以满足系统实时性。

3.13.2CAN 控制器的 TTCAN

HC32F4A0 系列的 CAN 控制器，在硬件上支持 TTCAN 的 Level1，不支持时间主节点（Time Master）功能。

3.13.2.1 TTCAN 时序

CAN 控制器的 TTCAN 通过寄存器 SBT 和 TTCFG 的 T_PRESC 位来设定网络时间单元（Network Time Unit，NTU）。寄存器位 TTCFG.T_PRESC 用于对 SBT 设定的位时间进行分频，分频数为 1、2、4 和 8。如果 SBT 设定的位时间为 t，TTCFG.T_PRESC 指定的分频数为 N，那么 NTU 则为 Nt。CAN 控制器的本地时钟（Local Time）、发送使能窗口（Transmit Enable Window，寄存器位 TRG_CFG.TEW）、触发时间寄存器（TT_TRIG）和触发看门时间寄存器（TT_WTRIG）都以 NTU 作为基本计时单位。

当 CAN 控制器的 TTCAN 功能使能（TTCFG.TTEN=1）后，16 位计数器（本地时钟 Local Time，计数值为 cycle time）开始计数，当参考消息被成功接收时，CAN 控制器将 Sync_Mark 拷贝给 Ref_Mark，Ref_Mark 将 cycle time 设置为 0。成功接收参考消息置位 RTIF.RIF 标志，而成功发送参考消息置位 RTIF.TPIF 标志或者 RTIF.TSIF 标志。此时需要准备下一个动作的触发条件。

触发条件可以是接收触发。该触发仅触发中断可用于检测期待的消息没有被收到。

触发条件也可以是发送触发。该触发开始发送通过寄存器位 TRG_CFG.TTPTR 指定的 TB SLOT 里的数据。如果选定的 TB SLOT 被标记为空，则不开始发送，但置位中断标志。

CAN 控制器的驱动程序，提供了一个结构体用于配置 TTCAN（如程序清单 3-3），其中就有用于配置 TTCAN 时序的参数。

```
typedef struct
{
    uint8_t u8TransBufMode;
    uint8_t u8NTUPrescaler;
    uint32_t u32RefMsgIDE;
    uint32_t u32RefMsgID;
    uint16_t u16TrigType;
    uint16_t u16TxEnWindow;
```

```
    uint16_t u16TxTrigTime;
    uint16_t u16WatchTrigTime;
} stc_can_ttc_cfg_t;
```

程序清单 3-3 CAN 控制器的 TTCAN 配置结构体

其中：

- u8TransBufMode：发送 Buffer 模式（详见 3.12.2.2 节）；
- u8NTUPrescaler：SBT 设定的位时间的分频数；
- u32RefMsgIDE：参考消息的 ID 扩展位；
- u32RefMsgID：参考消息的 ID；
- u16TrigType：发送触发类型；
- u16TxEnWindow：发送使能窗口时间（单位 NTU），该参数指定的窗口时间即为独占窗；
- u16TxTrigTime：发送触发时间（单位 NTU），从收到参考消息开始，等待该参数指定的时间后，将数据发出；
- u16WatchTrigTime：看门触发时间（单位 NTU），在该参数指定的时间内，如果未收到参考消息，相应标志位置位。

3.13.2.2 TTCAN 模式下的 TBUF 模式

寄存器位 TCTRL.TTTBM 为 1 时，PTB 和 STB SLOT 一样组成 TB SLOT，通过寄存器位 TBSLOT.TBPTR 指定发送 BUF（发送 BUF 被 TBSLOT.TBPTR 指定后，才可写入数据）。寄存器位 TBSLOT.TBPTR 为 0 时，指向 PTB，为 1 时指向 STB SLOT1，依次类推。发送 BUF 写入数据后，通过寄存器位 TBSLOT.TBF 标记指定的发送 BUF 已填充数据，待发送。此时寄存器位 TCMD.TBSEL 和 TCTRL.TSNEXT 无任何意义而可以被忽略。

寄存器位 TCTRL.TTTBM 为 1 时，PTB 不具有任何特殊的属性，和 STB SLOT 一样，传送完成标志也采用 TSIF；发送 BUF 没有 FIFO 模式和优先级仲裁模式，同时也只有一个选定的 SLOT 可以发送数据；传输开始需要采用时间触发方式，寄存器位 TCMD.TPE、TCMD.TSONE、TCMD.TSALL、TCMD.TPA 和 CFG_STAT.TPSS 被固定为 0 且被忽略。

寄存器位 TCTRL.TTTBM 为 0 时，组合使用事件驱动通信和接收时间戳功能。在该模式下，PTB 和 STB 的功能和 TTCAN 功能禁止（TTCFG.TTEN=0）时一致，因此 PTB 始终具有最高的优先级，而 STB 可以工作在 FIFO 模式或者仲裁模式。

3.13.2.3 TTCAN 触发方式

通过寄存器 TRG_CFG.TTYPE 定义 TTCAN 的触发方式，TRG_CFG.TTPTR 指定发送 SLOT，而 TT_TRIG 指定触发器的触发时间，即触发时的 cycle time 的值。

TTCAN 有以下五种触发方式。

立即触发。通过写 TT_TRIG 的高位（不在意写入的值），启动触发器发送数据，与是否收到参考消息无关。此模式下，TRG_CFG.TTPTR 选定的 TBUF SLOT 内的数据会立即发送。TTCFG.TTIF 不置位。

时间触发。如果一个节点期待在特定的时间窗口内（由寄存器 TT_TRIG 指定）收到期待的数据，则可以使用时间触发方式。如果 TT_TRIG 值小于实际的 cycle time（本地时钟的计数值），则 TTCFG.TEIF 置位且无其它动作。时间触发方式仅通过置位 TTCFG.TTIF 标志产生中断，并无其他功能。

单次发送触发。单次发送触发方式用于在执行时间窗口内（通过 TRG_CFG.TEW 位设定 ISO11898-4 规定的 1~16 个 NTU）发送数据。此时，忽略 CFG_STAT.TSSS 位。将数据写入发送 BUF 后，通过写 TT_TRIG 的高位（不在意写入的值），启动触发器，在本地时钟计数到与 TT_TRIG 相匹配时，开始发送数据。如果在规定的发送使能时间窗口（TRG_CFG.TEW）内数据没有开始发送，则帧被丢弃。对应的发送 BUF SLOT 被标记为空，并且置位 RTIF.AIF，对应的发送 BUF 内的数据不会被改写，因为可以通过置位 TBSLOT.TBF 再次发送。如果 TT_TRIG 值小于实际的 cycle time，则 TTCFG.TEIF 置位且无其它动作。

发送开始触发。发送开始触发方式用于仲裁时间窗口内，参与仲裁。CFG_STAT.TSSS 用于决定是否自动重发或者单次发送模式。如果 TRG_CFG.TTPTR 寄存器指定的消息没有被成功发送，可以使用发送停止触发来停止该发送。如果 TT_TRIG 值小于实际的 cycle time，则 TTCFG.TEIF 置位且无其它动作。

发送停止触发。发送停止触发用于停止通过发送开始触发已经开始的发送。如果仲裁失败或者有错误，可以用发送停止触发去停止这个发送。假如在 cycle time=0x100 时发送开始，期待在 cycle time=0x200 时该帧一定会（无仲裁失败，无错误）发送完成，此时设定停止触发 cycle time 为 0x200（设置寄存器 TT_TRIG 的值为 0x200），将触发方式设置为发送停止触发，到时间就停止，可以将冲裁失败的或有错误的发送停止掉。如果发送被停止，则发送帧被舍弃，置位 RTIF.AIF 并将选定的 TBUF SLOT 标记为空，但 TBUF SLOT 内的数据不会被改写，可以通过置位 TBSLOT.TBF 就可以再次发送。如果 TT_TRIG 值小于实际的 cycle time 则 TTCFG.TEIF 置位且执行停止。

除了立即触发方式外，所有的触发器都使用 TTCFG.TTIF 标志。TCTRL.TTTBM 为 1 时，只支持时间触发方式。这几种触发方式可根据应用灵活切换。

3.13.3CAN 控制器的中断

CAN 控制器提供如表 3-2 所示的中断类型。

中断标志	描述
RIF	接收中断
ROIF	接收上溢中断
ROIF	接收BUF满中断
RAFIF	接收BUF将满中断
TPIF	PTB发送中断
TSIF	STB发送中断
EIF	错误中断
AIF	取消发送中断
EPIE	错误被动中断
ALIF	仲裁失败中断
BEIF	总线错误中断
WTIF	触发看门中断
TEIF	触发错误中断
TTIF	时间触发中断

表 3-2 CAN 控制器的中断类型

CAN1 对应的中断源常量为 INT_CAN1_HOST（数值 340），CAN2 对应的中断源常量为 INT_CAN2_HOST（数值 341），CAN1 和 CAN2 可使用的独立中断向量为 Int000_IRQn~ Int031_IRQn，Int092_IRQn~ Int097_IRQn，可使用的共享中断向量为 Int138_IRQn。

4 CAN 控制器的应用流程

4.1.1 CAN2.0 和 CAN-FD 的初始化配置

CAN 控制器的 CAN2.0 和 CAN-FD 初始化配置流程如图 4-1 所示的流程图。

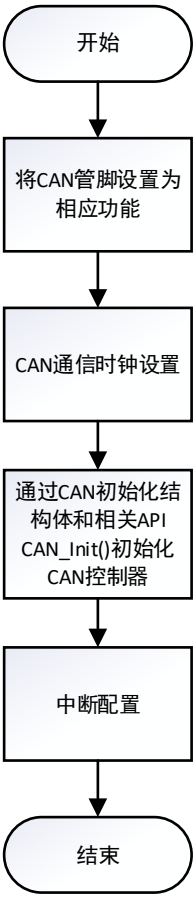

图 4-1 CAN2.0 和 CAN-FD 初始化配置流程图

其中，主要的初始化参数在 CAN 控制器的初始化结构体中，如程序清单 4-1 所示。

```
typedef struct
{
    uint8_t u8WorkMode;
    stc_can_bt_cfg_t stcSBT;
    en_functional_state_t enCANFDCmd;
    stc_can_fd_cfg_t stcFDCfg;
    uint8_t u8TransMode;
    uint8_t u8STBPrioMode;
    uint8_t u8RBSWarnLimit;
    uint8_t u8ErrWarnLimit;
    stc_can_af_cfg_t *pstcAFCfg;
    uint16_t u16AFSel;
```

```
    uint8_t u8RBStoreSel;
    uint8_t u8RBOvfOp;
    uint8_t u8SelfACKCmd;
} stc_can_init_t;
```

程序清单 4-1 CAN 初始化结构体

参数说明如下：

- u8WorkMode：用于指定 3.2.3 节所描述的工作模式；
- stcSBT：CAN2.0 位速率（波特率）配置结构体；
- enCANFDCmd：CAN-FD 功能开关；
- stcFDCfg：CAN-FD 配置结构体，包括位速率及延迟补偿的配置；
- u8TransMode：传输模式，包括：PTB 和 STB 自动重发模式、PTB 单次发送 STB 自动重发、PTB 自动重发 STB 单次发送、PTB 和 STB 单次发送；
- u8STBPrioMode：指定 STB 优先级模式：FIFO 模式（先写入先发送）或优先级仲裁模式（ID 小的优先发送）；
- u8RBSWarnLimit：当接收缓冲器接收到的帧的个数大于或等于该参数指定的个数（1~8）后，标志位 RTIF.RAFIF 置位，并且接收将满中断产生（若已使能）；
- u8ErrWarnLimit：指定错误警告数，参数范围 0~15，实际的错误警告数是(u8ErrWarnLimit +1) * 8；
- pstcAFCfg：指向接收筛选器配置结构体数组，用于批量设置接收筛选器；
- u16AFSel：批量选择需要使用的接收筛选器；
- u8RBStoreSel：指定接收缓冲器需要接收的数据，接收所有收到的数据还是只接收正确的数据；
- u8RBOvfOp：指定接收缓冲器溢出时的操作，丢弃新接收到的数据还是保持新接收到的数据，保持新接收到的数据时，最先收到的数据会被覆盖；
- u8SelfACKCmd：自动 ACK 功能开关，仅用于外部回环模式。

除了基本的初始化配置，还可根据需要配置中断。

4.1.2 TTCAN 的初始化配置

CAN 控制器的 TTCAN 初始化配置流程如图 4-2 所示的流程图，示例代码如程序清单 4-1 所示。

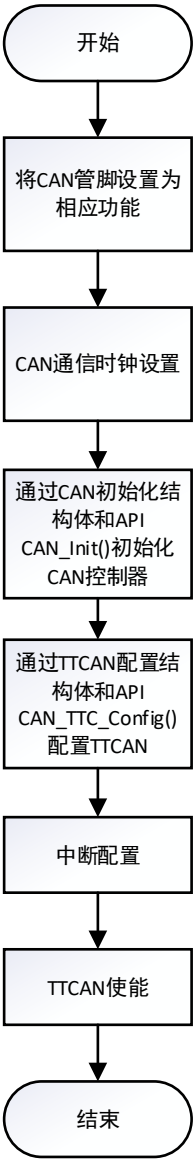

图 4-2 TTCAN 初始化配置流程图

```
static void CanConfig(void)
{
    stc_can_init_t stcInit;
    stc_can_ttc_cfg_t stcTTCCfg;
    /* 接收筛选器，按需配置 */
    stc_can_af_cfg_t astcAFCfg[] = { \
        {APP_CAN_AF1_ID, APP_CAN_AF1_ID_MSK, APP_CAN_AF1_MSK_TYPE}, \
        {APP_CAN_AF2_ID, APP_CAN_AF2_ID_MSK, APP_CAN_AF2_MSK_TYPE}, \
        {APP_CAN_AF3_ID, APP_CAN_AF3_ID_MSK, APP_CAN_AF3_MSK_TYPE}, \
        {APP_CAN_AF4_ID, APP_CAN_AF4_ID_MSK, APP_CAN_AF4_MSK_TYPE}, \
    };

    /* 将CAN控制器的引脚配置为相应功能。 */
    GPIO_SetFunc(APP_CAN_TX_PORT, APP_CAN_TX_PIN, \
                 APP_CAN_TX_PIN_FUNC, PIN_SUBFUNC_DISABLE);
```

```
    GPIO_SetFunc(APP_CAN_RX_PORT, APP_CAN_RX_PIN, \
                 APP_CAN_RX_PIN_FUNC, PIN_SUBFUNC_DISABLE);

    /* 配置CAN控制器的通信时钟。 */
    CLK_CAN_ClkConfig(APP_CAN_CLK_CH, APP_CAN_CLK_DIV);

    /* 指定CAN控制器初始化结构体的参数。 */
    (void)CAN_StructInit(&stcInit);
    stcInit.u8WorkMode = CAN_MODE_NORMAL;
    stcInit.pstcAFCfg  = astcAFCfg;
    stcInit.u8RBOvfOp  = CAN_RB_OVF_SAVE_NEW;
    stcInit.u16AFSel   = APP_CAN_AF_SEL;

    /* 配置波特率。根据通信时钟频率，设置合适的时钟分频。 */
    stcInit.stcSBT.u32Prescaler = 8U;
    PWC_Fcg1PeriphClockCmd(APP_CAN_PERIP_CLK, Enable);
    (void)CAN_Init(APP_CAN_UNIT, &stcInit);

    /* 配置TTCAN，包括NTU、参考消息的ID、触发方式等。 */
    (void)CAN_TTC_StructInit(&stcTTCCfg);
    /* NTU is slow bit time * 8, is 16us. */
    stcTTCCfg.u8NTUPrescaler   = CAN_TTC_NTU_PRESC_8;
    stcTTCCfg.u32RefMsgIDE     = APP_CAN_TTC_REF_IDE;
    stcTTCCfg.u32RefMsgID      = APP_CAN_TTC_REF_ID;
    stcTTCCfg.u16TrigType      = CAN_TTC_TRIG_SSHOT_TRANS_TRIG;
    stcTTCCfg.u16TxTrigTime    = 31250U;
    stcTTCCfg.u16WatchTrigTime = 65000U;
    CAN_TTC_ClrStatus(APP_CAN_UNIT, CAN_TTC_FLAG_ALL);
    (void)CAN_TTC_Config(APP_CAN_UNIT, &stcTTCCfg);

    /* 按需配置中断。 */
    CanIrqConfig();
    /* 使能TTCAN功能，本地时钟开始计时。 */
    CAN_TTC_Cmd(APP_CAN_UNIT, Enable);
}
```

程序清单 4-2 TTCAN 初始化配置

该初始化配置示例代码来自例程 can_ttc（Time-triggered Communication）。该例程配置如下：

1) 系统时钟设置为 240MHz；
2) CAN 的通信时钟选用系统时钟的 3 分频，频率 80MHz；
3) CAN 波特率 500Kbps，位时间为 2us；
4) TTCAN 的本地时钟为 CAN 波特率的 8 分频，即 NTU 为 16us；

TTCAN 的触发方式为单次发送触发，触发时间是 31250 个 NTU（500ms，即收到参考消息 500ms 后会将已经写入发送缓冲器的数据发出）。

5 总结

本篇应用笔记主要介绍了 HC32F4A0 系列 MCU 搭载的 CAN 控制器的功能及应用。在实际应用中，用户须结合 CAN 协议规格书和 HC32F4A0 的用户手册，对控制器做正确的配置及合理的应用。

6 版本信息 & 联系方式

日期	版本	修改记录
2021/7/27	Rev1.0	初版发布

如果您在购买与使用过程中有任何意见或建议，请随时与我们联系。

Email：mcu@hdsc.com.cn

网址：http://www.hdsc.com.cn/mcu.htm

通信地址：上海市浦东新区中科路 1867 号 A 座 10 层

邮编：201203