CAN Interface Layer Module Design Reference

Version	V0.0.1
Status	Release
Authors	Mingfen XIAO

1. Documents Information

1.1 Change Log

Version	Date	Status	Authors	Comments
V0.0.1	2018.09.26	Draft	Mingfen XIAO	1.Inital version

1.2 Reference Documents

Index	Documents Name	Version
1		
2		
3		
4		
5		
6		
7		

2. Contents

1. Documents Information 2
   1.1 Change Log 2
   1.2 Reference Documents 2
2. Contents 3
3. Introduction 4
   3.1 Architecture Overview 4
   3.2 Modules Files Structure 4
   3.3 Modules Description 5
4. Function Description 5
   4.1 Initialization Function 5
   4.2 Main Function 5
   4.3 Receive CAN Frame 6
   4.4 Send CAN Frame 7
   4.6 Software Filter 7
   4.7 DLC Check 7
   4.8 Timeout Check 8
   4.9 Timeout Behavior 8
5. API Description 9
   5.1 CanIf_Init 9
   5.2 CanIf_MainFunction 9
   5.3 CanIf_RecvInterruptCallback 9
   5.4 CanIf_SendInterruptCallback 10
6. Term 10

3.Introduction

3.1 Architecture Overview

图 3-1 为 Communication stack 的架构图。

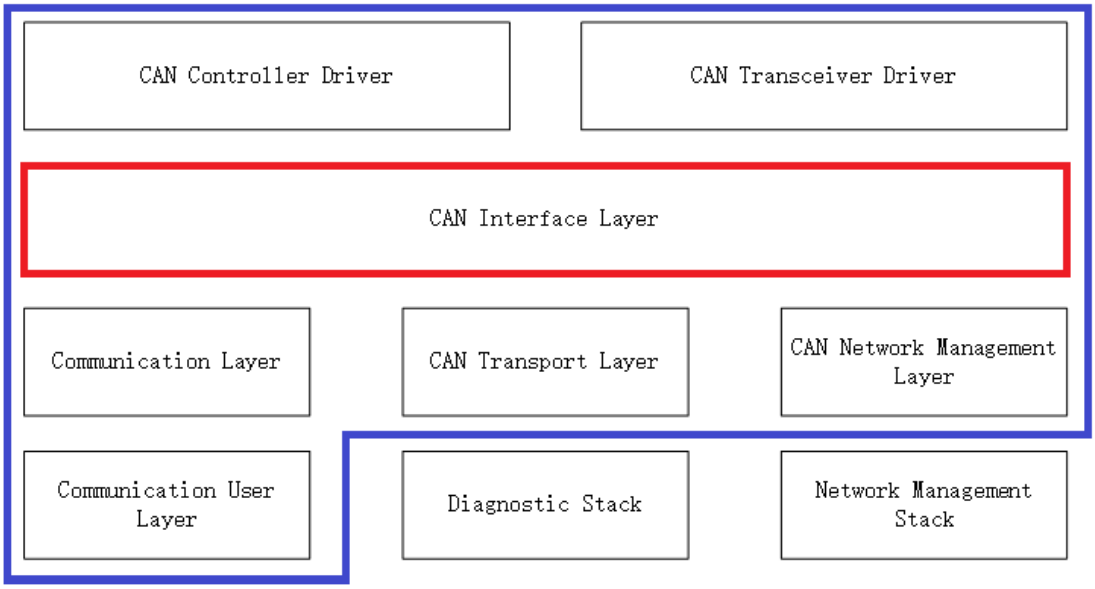

图 3-1 Communication Stack Architecture

如图 3-1 所示，蓝色区域为 Communication Stack 的各个模块。另外两个模块是和 Communication Stack 交互紧密的模块。

对于 Communication Stack 各个模块的功能和作用如下：

Can controller Driver Layer：对 CAN 控制器进行硬件的抽象以及驱动程序的实现。

Can Transceiver Driver Layer：对 CAN 收发器进行硬件的抽象以及驱动程序的实现。

Can Interface Layer：对硬件抽象层和服务层进行隔离。

Communication Layer：通信的服务层，主要对报文的编解码等操作。

Communication User Layer：通信的应用程序，主要是对信号的功能处理。

Can Transport Layer：诊断报文的处理,主要是对 TP 层参数进行配置和实现 TP 层的逻辑。

Can Network Management Layer：网络管理报文的处理，主要实现 CAN 网络网络的功能，但是对于不同总线的网络管理的功能不进行实现。

3.2 Modules Files Structure

File Name	File Type	Description
CanIf.c	Static	CanIf 模块的源文件。实现 CanIf 模块的函数
CanIf.h	Static	CanIf 模块的头文件。对 CanIf 模块的函数进行声明
CanIf_Type.h	Static	CanIf 模块数据类型的定义

VFB_CanIf.h	Dynamic	CanIf 需要使用外部模块的接口进行 API 的映射
CanIf_Cfg_Define.h	Dynamic	CanIf 模块用户功能开关选择以及参数的配置
CanIf_Cfg_Table.h	Dynamic	CanIf 模块用户列表的定义

3.3 Modules Description

CAN Interface Layer 是介于硬件抽象层（Can driver layer and Can transceiver layer）和服务层（can transport layer and communication layer,etc.）直接的一个接口层。目的是为了将硬件抽象层和服务层进行隔离，减小服务层以上的模块对硬件平台的依赖性。

通过这样的隔离，在 CanIf 层以上的模块将会硬件没有任何的关系，理论上可以将这些模块移植到任意的硬件平台上。

CanIf 模块的主要功能有以下几部分组成：

1. 对 CAN 控制器和收发器进行直接的控制，即只有在此模块中才会直接调用 CAN 控制器和收发器的驱动函数。
2. 对报文进行收发的操作。
3. 对于报文的接收，将实现 DLC 检测，timeout 的检测，以及软件过滤等功能。
4. 根据接收报文的 ID 将报文传送到不同的模块，比如 COM，CANTP，CANNM 等
5. 对于报文的发送，将实现周期型，事件型，混合型报文的发送。

4. Function Description

本章节会详细描述 CAN 接口模块具体的功能策略。

4.1 Initialization Function

1. 初始化发送控制信息。
2. 初始化接收控制信息。

4.2 Main Function

Main function 主要是实现 CanIf 对报文收发的处理过程。Main Function 有分为 CanIf_RxMainFunction 和 CanIf_TxMainFunction。

当前设计为 RX 和 TX 只能使用同一个执行周期，因为在当前的设计中将 CanIf_RxMainFunction 和 CanIf_TxMainFunction 设计为 static function 了。

4.3 Receive CAN Frame

对于报文的接收存在两种方式：中断和查询的方式。

当接收到报文之后，CanIf 将会调用 CAN Driver 的接口将 Hardware buffer 的数据读取出来，并且将 CAN 报文写入到 CanIf_CanRecvMsgBuff 中。但是在写入之前会对 CanIf_PreCopy 的返回接口进行判断。只有在 CanIf_PreCopy 返回 OK 的时候，数据才会写入到 CanIf_CanRecvMsgBuff 中。

CanIf_PreCopy 函数是给应用层来实现的，所以通过这个函数，应用层可以实现一些拓展的功能。

在 RX Main function 中将通过轮询的方式去查询 CanIf_CanRecvMsgBuff 是否存在新的数据。当 CanIf_CanRecvMsgBuff 存在新的数据的时候，就会启动相关的 check 功能，比如软件过滤器对 ID 进行 check，DLC check 等一系列的 check 功能。

当 Check 通过之后，需要将 CanIf_CanRecvMsgBuff 中的数据读取出来，根据 Message Id 来将 Message 的数据通过给上层。如果是应用报文则通知给 Com；如果是诊断报文则通过给 CANTP；如果是网络管理报文则通过给 CANNM。

RX 功能的配置列表为 CanIf_CanMsgRxList，主要是指定各个 check 是否使能，并且需要保证每一个接收到的信号都需要在此列表中。

对于是否接收到新报文是根据 CanIf_CanRecvMsgBuff 数组的读写指针来进行判断的，即把 CanIf_CanRecvMsgBuff 看着一个循环队列来进行操作。当有数据写入的时候，写指针向后移动一下，当数据读取出来的时候，读指针向后移动一下。所以当读写指针不一致的时候，即存在新的数据。

在这里有一个说明一下：为什么不将 can message 的数据存放在 CanIf_CanMsgRxList 数组中，而是存放在 CanIf_CanRecvMsgBuff 数组中呢？

因为在定义的时候就会对 CanIf_CanMsgRxList 数组进行初始化，并且设定相关的参数，所以每个 Message 存放的位置就是固定的。基于这种情况，如果从 CAN 控制器读取的新的报文的时候，都需要先查询 Message Id 存放的位置，在将 Message 的数据存在到 CanIf_CanMsgRxList 数组里面，当需要接收的报文比较多的时候，每次用于查询位置的时间就会比较多。当使用中断的方式来实现报文的接收的时候，位置的查询是在中断函数中执行的，这样的方式会影响 MCU 的整体性能，所以在此策略中设计了 CanIf_CanRecvMsgBuff 数组用于临时存在接收到的数据。

但是需要注意的时候，这个数据的大小需要根据实际接收报文的数量，报文的周期，TX main function 执行的周期来定义的。如果定义太小的话，主要引起的问题是因为 buffer 太小，导致在前面的报文还没有处理完成，后面来的报文就将前面的报文覆盖掉了，会引起丢帧的问题。

4.4 Send CAN Frame

对于报文的发送策略，在 CAN 控制器端可以使用中断和轮询的方式来进行发送。

对于 CanIf 而言，定义一个 CanIf_CanMsgTxList 和 CanIf_CanTxMsgCtrInfo 两个参数。

CanIf_CanMsgTxList：主要是定义报文类型，周期，对于事件型报文还有连续发送次数，以及报文的数据等参数。

CanIf_CanTxMsgCtrInfo：主要是控制 CanIf_CanMsgTxList 数组的 index 以及使用 can 控制器硬件缓冲区的 index。

CanIf 的发送策略为：

1. 遍历 CanIf_CanMsgTxList 数组，检测报文 ID 是否有效。
2. 检测此报文的类型。
3. 检测是否达到发送的时间。
4. 如果到达发送的时间，则调用 can 控制器的驱动函数，进行报文的发送。
5. 对于事件型和混合型报文，还存在连续发送次数的控制。

对于报文周期的控制，是通过计数的方式来实现的，counter = cycle timer / task tick.

4.6 Software Filter

对于软件过滤器，是对于全局设置的，不能对于单独的报文进行设置。此功能也称之为 Message Id Check.

当启用 Software Filter 功能之后，只有接收的报文存在 CanIf_CanMsgRxList 数组中，并且该报文的 valid flag 为 TURE 的时候，此报文才有效；否则，丢弃此报文。

4.7 DLC Check

对于每个报文都可以单独配置是否使能 DLC check 功能。

当启用 DLC check 功能之后，只有接收报文的 DLC 和配置的 DLC 一致的时候，报文才有效；否则丢弃此报文。

DLC Check 的结果会存放在 CanIf_CanMsgRxList 中，服务层可以通过相关的接口来查询 Check 的结果。

配置参数位于：CanIf_CanMsgRxList[index]. MsgCheckRet bit1

注：DLC Check 的前提是 Message Id Check 成功。

4.8 Timeout Check

对于每个报文都可以单独配置是否进行 timeout 的功能。需要配置两个参数：

1. Timeout 的使能开关；
2. timeout 的时间；

timeout check 的故障确认机制为：

当 CanIf 模块运行起来之后，就会启动 timeout 计数器的自减，当计数器达到 0 的时候，就视为 timeout。

在 CanIf 接收到报文的时候，就会重载 timeout 的计数器。

由于 CanIf 运行起来之后，就会进行 timeout 的检测。所以有一部分 ECU 的需求为：在系统启动一定时间之后在启动 timeout 的诊断。对于实现这种需求的解决方案为：在 DCM modules 中设置一个 diagnostic condition（将诊断启动的超时时间作为一个诊断启动的条件）。当设定 diagnostic condition 使能之后，这个时候才会产生 DTC。

timeout check 的恢复机制为：

只要接收到报文就视为 timeout 的恢复。

对于实现报文 timeout 确认和恢复不满足的需求，也可以和 DCM 模块联合起来实现。即设定一个 timeout 诊断的检测周期，每一个周期中读取一个 timeout 的状态，根据每次读到的 timeout 的状态实现一定的逻辑来确定相应的 DTC。

Timeout Check 的结果存放在 CanIf_CanMsgRxList 中，服务层可以通过相关的接口来查询 check 的结果。

配置的参数位于：CanIf_CanMsgRxList[index]. MsgCheckRet bit2

注：1. Timeout Check 的前提是 Message Id Check 成功。

4.9 Timeout Behavior

当出现 timeout 的行为之后，就会引出下面问题：

当出现 timeout 之后，上层读取通过 API 来读取信号的值，CanIf 应该返回什么值给上层？

在现在设计的方案中，此行为是可以进行配置的。有两种选择的方案。第一种：当出现 timeout 行为之后，信号值将更新为信号的初始值；第二种：当出现 timeout 行为之后，信号值将保持为最后结束到的值。此行为在 CanIf 中实现。

当出现 timeout 行为时，CanIf 会将 timeout check 的结果存放在 CanIf_CanMsgRxList 数组中，并且根据用户配置的行为来做相应的操作。当用户配置的行为为信号值更新为初始值时，CanIf 会将 init 的值主动发送给 Com modules.

配置的参数位于：CanIf_CanMsgRxList[index]. MsgCheckRet bit0

注：Timeout Check 的前提是 Message Id Check 成功。

5.API Description

此章节主要描述 CanIf 向其他模块提供的接口。

5.1 CanIf_Init

Function Prototype	void CanIf_Init(void)
Parameters List	
Return Value	
Function Description	
Limitations	
Note	

5.2 CanIf_MainFunction

Function Prototype	void CanIf_MainFunction(void)
Parameters List	
Return Value	
Function Description	
Limitations	
Note	

5.3 CanIf_RecvInterruptCallback

Function Prototype	uint8 CanIf_RecvInterruptCallback(uint8 ChNo)
Parameters List	
Return Value	
Function Description	
Limitations	
Note	

5.4 CanIf_SendInterruptCallback

Function Prototype	uint8 CanIf_SendInterruptCallback (uint8 ChNo)
Parameters List	
Return Value	
Function Description	
Limitations	
Note	

6. Term

Term	Description
API	Application Programming Interface
CAN	Controller Area Network