Communication 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	2018.11.15	Draft	Mingfen XIAO	1.Inital version

1.2 Reference Documents

Index	Documents Name	Version
1		
2		
3		
4		
5		
6		
7		

2. Contents

3.Introduction

3.1 Architecture Overview

图 3-1 为 Communication stack 的架构图。

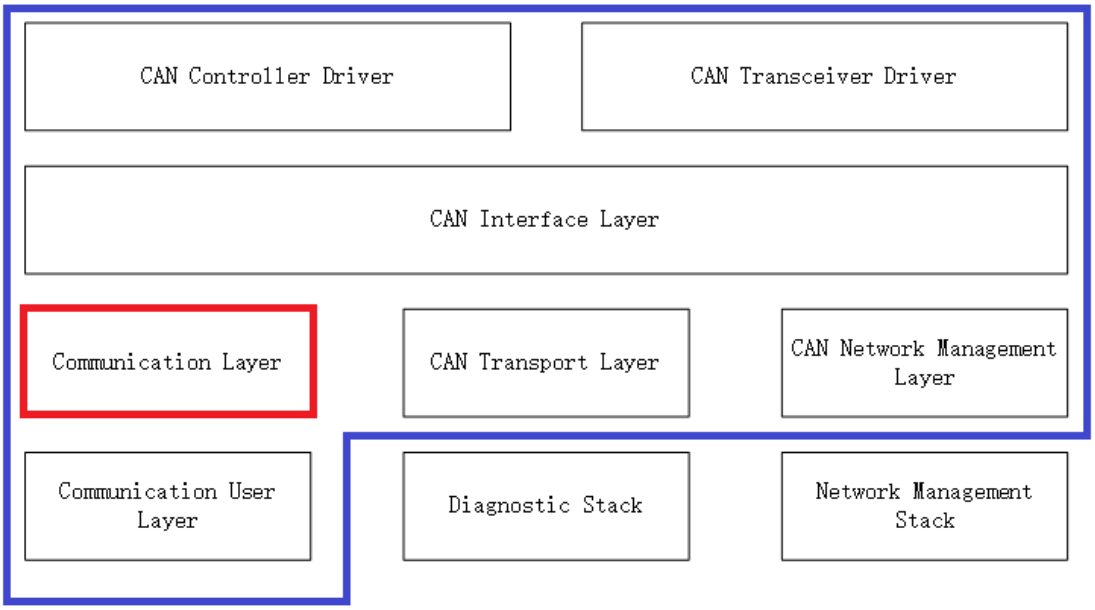

图 3-1 Communication Stack Architecture

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
Com.c	Static	Com 模块的源文件。实现 Com 模块的函数
Com.h	Static	Com 模块的头文件。对 Com 模块的函数进行声明
Com_Type.h	Static	Com 模块数据类型的定义

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

3.3 Modules Description

Communication Layer 是通信的服务层，是介于应用层（ComUser）和接口层（CanIf）之间的一个处理协议栈的服务层。

Communication Layer 的主要的作用是为了减少软件对硬件的依赖性，实际上 com module 是应该适用所有的总线报文的处理。但是在现阶段，主要是匹配 CAN message 协议栈的处理。最终的目标是为了实现匹配所有的总线。

在 AUTOSAR 架构中，从 BusIf 以上的模块，应该能够适用各个硬件平台的移植。所以 Com Module 理论上应该适用于各种类型总线报文的协议处理。

Com 模块主要功能有以下几个部分组成：

1. 接收来自 BusIf 层的报文消息；
2. 对接收报文进行解析；
3. 对发送报文进行组装；
4. 接收应用层的信号；
5. 向 CANIF 填充报文进行报文的发送；
6. 获取 CANIF 中 check 的相关状态；
7. 为应用层提供 read/write signal 的 API。

4.Function Description

4.1 Initialization Function

主要是对 Com_BusTxMsgList 和 Com_BusTxMsgList 两个数组进行初始化的操作。

对于 Com_BusTxMsgList，在初始化的时候，需要将 BusIf 的 TX buffer 和 Com_BusTxMsgList 进行同步，保证两个 buffer 的数据一致。

对于 Com_BusRxMsgList，在初始化的时候，需要将 BusIf 的 RX buffer 和 Com_BusRxMsgList 进行同步，保证两个 buffer 的数据一致。对于接收的 buffer，主要是保证和 init 值保持一致即可。

----此部分在代码中还没有实现

其实对于初始化的部分，还有一个方案，就是通过配置工具，在创建 BusIf 和 com buffer

的数组的时候，保证初始值一致。

-----因为暂时还没有去实现配置工具的开发，所以在现阶段将按照第一个方案执行。

4.2 Main Function

在现在的设计架构中，设计了 Com_RxMainFunction 和 Com_TxMainFunction，用于实现对接收报文和发送报文协议的处理。

在现阶段的功能实现中，没有使用 Com_RxMainFunction 函数，所以 Com_RxMainFunction 函数也没有进行实现，只是作为预留函数。

Com_TxMainFunction 函数主要实现发送相关的功能。

在现阶段的功能实现中，会在 Main function 中使用 polling 的方式去查询 Com_BusTxMsgList 数组是否存在 signal value 的更新。如果检测到了 signal 更新，则 COM 会会将 TX buffer 的数据更新到 CanIf 中的 TX buffer 中。

4.3 Receive Message

Com 是通过 RecvNotificationFunction 来获取 BusIf 层的总线报文。获取到总线报文之后，就存放在 Com_BusRxMsgList 中。

当数据存放到 Com_BusRxMsgList 中之后，就等待应用层来读取信号了。

4.4 Send Message

在 com 模块中，对于报文的发送存在两种方式：一种为立即发送，第二种为轮询的发送。

立即发送，当应用层需要发送报文时，调用对应的 API。在 COM 中实现的功能为：

1. 基于 MsgId 和 ChNo 读取 BusIf 的 TX buffer 中的数据；
2. 基于读取到的数据，和需要更新的数据进行编码；
3. 将编码完成的数据，更新到 BusIf 的 TX buffer 中。

轮询发送，当应用层需要发送报文时，调用对于的 API。在 COM 中实现的功能为：

1. 将应用层需要更新的信号更新到 Com_BusTxMsgList 中。在此过程，就会完成数据的更新和编码；
2. 并且将 Com_BusTxMsgList 中对应的 update flag 更新；
3. 在 COM TX Main function 中周期的查询 update flag 是否更新；
4. 查询到 update flag 更新之后，就将 Com_BusTxMsgList 中的数据更新到 BusIf 的 TX buffer 中。

对于上面描述的两种方案，需要注意一下几点：

1. 由于使用第一种方式，每次更新数据的时候，都会从 BusIf 中读取数据；使用第二种方式，只有在初始化的时候，才会从 BusIf 中更新数据以保证同步，并且 BusIf 数据的来源为 com。所以对于同一个 ID 的报文，**不允许两种方式混合使用**。
2. 使用第一种方案，应用层发起更新数据的请求，到数据更新到 BusIf 模块的 TX buffer 中，时间差仅仅为函数执行的过程。
3. 使用第二种方案，应用层发起更新数据的请求，数据先更新到 Com_BusTxMsgList 中，之后在通过 polling 的方式向 BusIf 中更新。所以发起请求到数据更新到 BusIf 模块的 TX buffer 中，最大的时间差为一个 COM main 的周期。

----后期将考虑两种方式混合使用的策略。

4.5 Read Signal

此功能主要是对接收报文进行解码的，提供给应用层来读取信号的值。对于此功能，需要在应用模块的 VFB 层进行信号的封装。当调用封装好的 API 的时候，就可以直接读取到信号值。

在静态代码中主要是考虑 read signal 的策略。如果需要对 signal group 进行读取的话，可以在 VFB 层进行一次封装。这部分的策略，将在后续的配置工具中实现。

4.6 Write Signal

此功能主要是对发送报文进行编码的过程，提供给应用层来设置信号的值。对于此功能，需要在应用模块的 VFB 层进行信号的封装。当调用封装好的 API 的时候，就可以直接设置信号值。对于章节 4.4 中关于发送报文的策略，对应两个 write signal 的 API，分别为 Com_WriteTxSignal 和 Com_ImmediatelyWriteTxSignal。

在静态代码中主要是考虑 write signal 的策略。如果需要对 signal group 进行写的话，可以在 VFB 层进行一次封装。这部分的策略，将在后续的配置工具中实现。

4.7 Timeout

Timeout 在现阶段是设计了接收报文的 timeout 策略，对于发送报文的 timeout 策略，暂时不考虑。

Timeout 在 BusIf 层来进行 check 的，并且 check 的结果在存放在 BusIf 层。在 Com 层对于 Timeout 的处理，就是读取 BusIf 层的 check 结果。

当前的策略为，当应用层调用 read signal 的 API 来获取信号的时候，调用 BusIf 的 API 来获取 timeout 的状态，并且通过返回值的方式给到应用层。

[注意]：

1. 由于报文是从底层上传上来的，应用层在获取到 timeout 状态的前提是有对应的报文存储在 Com 的 RX buffer 中。在 BusIf 中接收到报文或者检测到报文丢失都向 Com 传送报文消息。
2. 当 read signal 的函数返回为 E_RET_NOT_FOUND 时，有多种可能的情况。第一种为系统不支持接收此报文；第二种情况为系统允许接收此报文，并且一上电总线就没有出现此报文，并且还没有到达 timeout 的时间点；第三种情况为 Com 定义 RX list 的长度过小，导致部分报文溢出（建议 Com 中 RX list 的大小应该大于或者等于 BusIf 的 RX list 的大小）。

5.API Description

5.1 Com_Init

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

5.2 Com_MainFunction

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

5.3 Com_RxMainFunction

Function Prototype	void Com_RxMainFunction (void)
Parameters List	
Return Value	
Function Description	

Limitations	
Note	

5.4 Com_TxMainFunction

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

5.6 Com_ReadRxSignal

Function Prototype	uint8 Com_ReadRxSignal(uint8 FormatType,uint8 ChNo,uint32 MsgId,uint8 StartBit,uint8 Length,uint8 *ptr_SignalValue)
Parameters List	FormatType： the can message format. Intel format : 0x00; Motorola format : 0x01 ChNo： MsgId： StartBit： Length： ptr_SignalValue
Return Value	E_MSG_TIMEOUT(0x0A) : indication the message is timeout E_RET_NOT_FOUND(0x06)： not find the message in the com rx buffer.
Function Description	
Limitations	
Note	

5.7 Com_WriteTxSignal

Function Prototype	Com_WriteTxSignal(uint8 FormatType,uint8 ChNo,uint32 MsgId,uint8 StartBit,uint8 Length,uint8 *ptr_SignalValue)
Parameters List	
Return Value	

Function Description	
Limitations	
Note	

5.8 Com_ImmediatelyWriteTxSignal

Function Prototype	Com_ImmediatelyWriteTxSignal(uint8 FormatType,uint8 ChNo,uint32 MsgId,uint8 StartBit,uint8 Length,uint8 *ptr_SignalValue)
Parameters List	
Return Value	
Function Description	
Limitations	
Note	

6.Term

Term	Description
API	Application Programming Interface
CAN	Controller Area Network