第3章 自动驾驶汽车软件平台

随着目前科学技术水平的不断提高，嵌入式系统已在各大企业以及高校中得到广泛应用。操作系统是应用软件的基础开发平台，在嵌入式系统应用的开发过程中，它不仅能够简化软件的开发环节，而且可以降低后续维护成本，所以逐渐成为嵌入式系统中不可或缺的一部分。由于汽车、工业控制以及航空航天等领域对时间响应有着明确的规定，嵌入式实时操作系统已潜移默化地应用于上述领域。

汽车电子领域的软件大多数是嵌入式软件。由于各类硬件产品种类的不一致性，以及整体嵌入式系统软件的快速发展，现阶段软件设计开发以封闭式为主。这样有利于开发指定设计的软件系统，例如针对特定硬件以及高效利用硬件资源等。这种涉及指定硬件和应用而开发的软件系统，其对硬件资源的高效应用以及软件本身的处理效率是十分可观的。

作为中国制造“2025”和“两化融合”战略中的重要举措，智能汽车引领着汽车产业转型升级的新方向。一汽集团和奇瑞公司等众多车企一致发布智能汽车发展战略，强调了软件技术和产品的重要性。在这期间，软件和互联网公司以跨界合作的方式加速了智能汽车的发展。阿里巴巴与上汽集团紧密合作，以 10 亿元人民币合资基金建设互联智能汽车平台。此外，国内数家互联网公司以线车载诊断系统（OBD）或车载设备为突破口进入智能汽车领域。上述例子可以表明，软件已逐渐成为汽车智能网联化和信息化发展的基础与核心。

3.1 软件平台概述

在汽车领域，人们对电子化的要求逐渐变高，汽车电子成本占据整车费用的份额越来越高，汽车上因使用了大量的电子控制单元（Electronic Control Unit，ECU），使得电子软件的开发难度越来越大。传统汽车的硬件部分占整车费用的 90%，而软件仅占 10%；但智能汽车的硬件部分占据整车费用的 40%，软件占 40%，内容占 20%。相关数据显示，目前的中高端汽车，汽车电子占整车成本已超过 30%，一辆智能汽车大约装备有 50～100 个 ECU，2 亿行左右的源代码，代码量与空客 A380 客机相当。约有 90%的智能汽车创新是通过汽车电子来实现的，其中，有 80%的创新取决于软件。软件创新逐渐胜过传统的硬件创新。众多汽车厂商

利用软件系统控制汽车架构，除了可以实现创新升级的优点之外，还可以很大程度地降低汽车重量和生产成本。但是，ECU 应用软件的开发对硬件平台的需求较高，这样导致了应用程序可移植性差、各软件模块不兼容等诸多问题。因此，汽车电子行业急需一个具有统一标准接口的嵌入式实时操作系统作为基础开发平台，能在任何类型的 ECU 中使用，将应用软件和底层硬件细节隔离开来，实现不同制造商提供的软件模块能够无缝集成。

为了实现容错操作的系统，汽车制造商必须从汽车的整体架构入手，确保将备援机制等安全措施部署在以下 3 个功能区：感测、运算、制动。这些区块之间以及与其他汽车、基础设施或云端等车外世界之间的所有互动（亦即联机）必须及时且可靠。作为自动驾驶的关键技术之一，它能实现很多的增值服务，其中包括远程诊断、软件更新、交通信息、过路费控管及付款等多种服务。但是，这些功能也增加了网络攻击的风险。因此，车载系统急需提供更多的安全功能。多层式策略通常是解决这个安全难题的关键。首先，在应用层执行相关安全通信，加上应用访问控制，确保流动数据的可信度，这对于无线软件更新（OTA）或相关服务功能特别重要。其次，软件平台的安全模块需要包含分隔化、入侵预防和 ECU 验证。最后，在硬件层方面，CPU 安全、针对整合硬件安全模块（HSM）的支持以及 ECU 自身的防火墙，这些功能都有利于为汽车建立全方位的安全平台。

越来越多的汽车制造商和系统供应商为了紧跟时代的步伐，逐步扩大功能。有些厂商愿意与服务供应商成为合作伙伴；有些则并购整间公司，掌握其先进的技术，尤其是软件方面的技术。这些事件证明，汽车产业越来越重视新技术和新方法的需求，由此带来的合作空间是巨大的。长远来看，采用开放式平台/开放式系统策略有助于加快整体创新流程，促进业界开发可互通、模块化且可扩充的自动驾驶系统。

近年来，我国智能汽车的软件研发技术得到高速发展。在车控软件方面，“核高基”重大专项部署支持开发实时嵌入式操作系统及开发环境、汽车电子控制器嵌入式软件开发平台和国产汽车电子基础软件平台等产品。国内软件平台厂商参照 OSEK/VDX 和 AUTOSAR 等国际标准，已经研发出面向 ECU 的操作系统产品及解决方案。浙江大学 ESE 实验中心成功开发出符合 AUTOSAR 标准的集成 ECU 开发工具链，支持用于 ECU 软件架构、网络系统配置、基础软件配置、诊断、标定和仿真测试的快速迭代开发模式。在车载软件方面，国内的阿里云研发出可用于车载终端的阿里云操作系统，且涌现出科大讯飞、高德等实力较强的车载应用软件供应商和普华、博思等车载终端系统供应商，这些厂商推出的相关产品在一些车型中已得到应用。从宏观角度来看，对比世界汽车产业发达的国家加速布局智能汽车，我国相关软件的发展步伐明显滞后，机遇与挑战并存。

为赶超发达国家，我国需要推进以深度融合的工作机制来推动软件技术和汽车行业的发展，结合软件产业和汽车行业的优势资源，制订并实施软件技术与汽车产业融合发展行动计划；需要大力支持本土软件企业与国产汽车厂商的深度合作，围绕汽车产业链关键环节进行联合攻关和协同创新；鼓励和带领国产汽车厂商采用软件平台技术，发展基于新一代信息技术的智能车联网技术和服务，提升汽车性能以及智能化水平。

3.2 自动驾驶汽车软件架构

3.2.1 AUTOSAR 软件构架

本节着重介绍 AUTOSAR 软件构架。对自动驾驶汽车软件构架来说，AUTOSAR 是

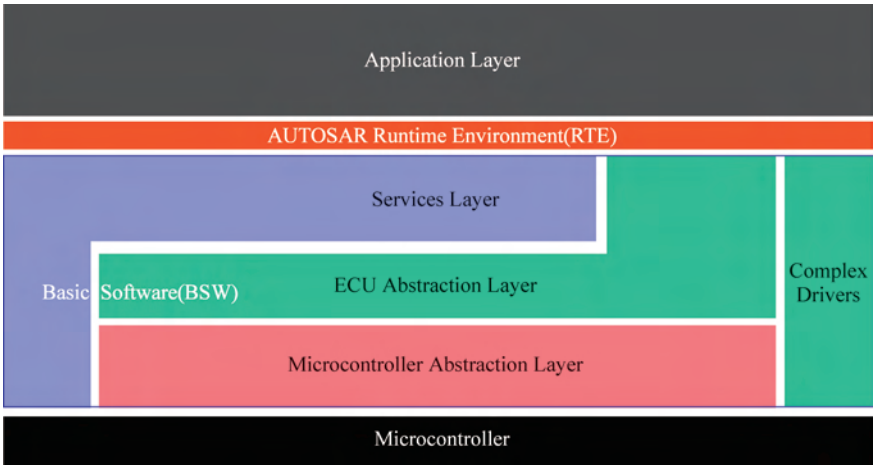
被大多数用户所熟知的一个开放的、标准化的软件构架。本节将详细介绍 AUTOSAR 的概念、国内外发展现状及 AUTOSAR 整体框架。

1. AUTOSAR 概述

AUTOSAR 是 Automotive Open System Architecture(汽车开放系统架构)的缩写,是一家专注于制定汽车电子软件标准的联盟。AUTOSAR 是由全球汽车制造商、零部件供应商及其他电子、半导体和软件系统公司联合建立,各成员企业之间保持开发合作伙伴关系。自 2003 年起,各伙伴公司携手合作,致力于为汽车工业开发一个开放的、标准化的软件架构。AUTOSAR 软件架构有利于车辆电子系统软件的交换与更新,并为高效管理愈来愈复杂的车辆电子、软件系统提供一个平台。AUTOSAR 在确保产品及服务质量的同时,提高了成本效率。

整车软件系统可通过 AUTOSAR 架构对车载网络、系统内存及总线的诊断功能进行深度管理,并改善了系统的可靠性和稳定性。目前支持 AUTOSAR 标准的工具和软件供应商都已经推出相应的产品,提供需求管理、系统描述、软件构建算法模型验证、软件构建算法建模、软件构建代码生成、RTE(Real Time Environment)生成、ECU 配置以及基础软件和操作系统等服务,帮助软件系统提供商实现无缝的系统软件架构开发流程。

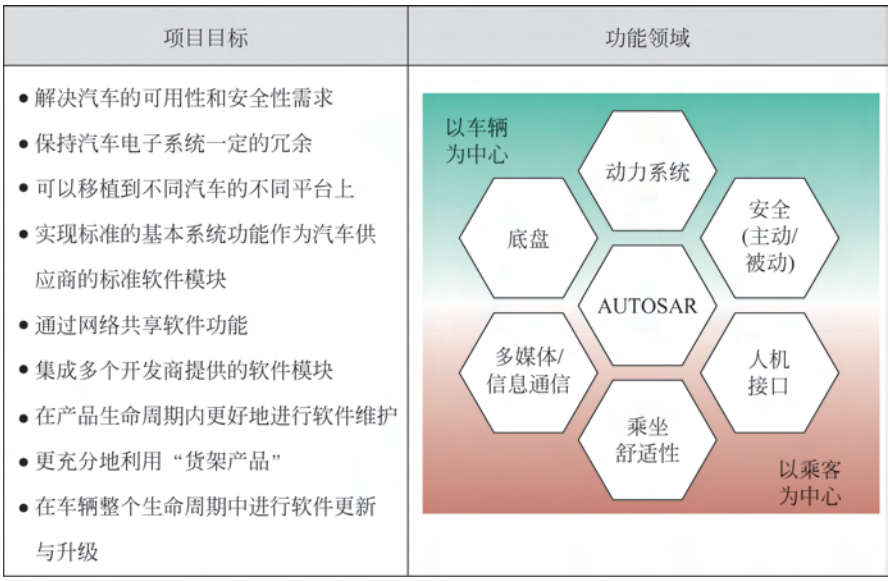
传统的 ECU 架构有以下两个缺点:抽象程度低;基础软件模块少。针对以上问题,AUTOSAR 规范提出了抽象程度更高的解决措施,划分出更多的基础模块。为了实现应用软件和硬件模块的解耦,汽车电子软件架构被抽象成 4 层,如图 3.1 所示,由上至下依次为:应用层(Application Layer)、运行时环境(Run Time Environment,RTE)、基础软件层(Basic Software,BSW)以及微控制器(Microcontroller)。应用层完全独立于硬件,只有基础软件层与硬件相关,而 RTE 实现这两者的隔离。这样,一方面,厂商可专注于开发特定的、有竞争力的应用层软件(位于 RTE 之上),另一方面,它使厂商所不关心的基础软件层(位于 RTE 之下)得到标准化。



■图 3.1 AUTOSAR 层面图

作为汽车电子行业的新兴标准,国内外对 AUTOSAR 规范的研究成为热点,并一致选择将原有符合 OSEK/VDX 规范的操作系统平滑升级至符合 AUTOSAR 规范的版本。面

对 AUTOSAR 规范正逐步取代 OSEK/VDX 规范的趋势，国内业界急需对 AUTOSAR 操作系统规范进行深入研究。AUTOSAR 组织规定的目标以及它所囊括的功能领域如图 3.2 所示。



■图 3.2　AUTOSAR 功能图

为了达到上述目标，针对在汽车电子行业中遇到的难题，AUTOSAR 采用的解决方案及其优点可以概述如表 3.1 所示。

表 3.1　AUTOSAR 面临的挑战及其优点

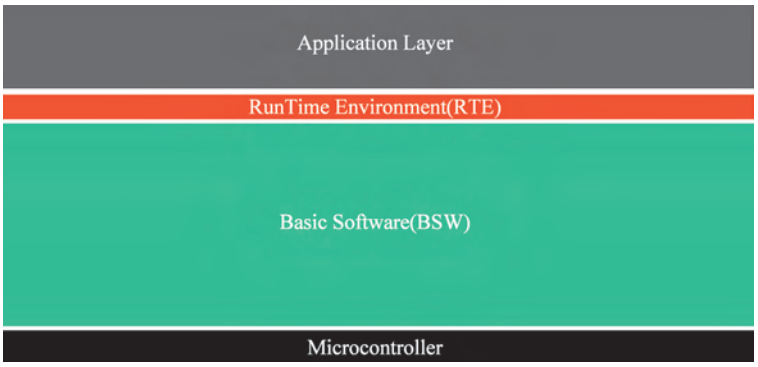
非 AUTOSAR 面临的挑战	AUTOSAR 方案	AUTOSAR 优势
对功能需求缺乏追溯手段，没有兼容性的工具链	需求交互格式标准化(ARXML)	从内容和格式上改进了规范，为无缝的工具链提供可能
基础软件模块不能复用，带来的时间和精力浪费	基础软件(BSW)	提高软件质量，供应商提供基础软件
升级主芯片带来大量的重新设计	微控制器抽象层(MCAL)	主芯片可以随意替换，MCAL 有芯片厂家提供
集成工作反锁	运行时环境(RTE)	集成自动化
软件耦合性大	接口标准化	不同供应商可交互组件

AUTOSAR 架构是为了改善汽车电子系统软件的更新与交换，同时更快捷有效地管理日趋复杂的汽车电子软件系统。AUTOSAR 规范的使用让不同结构的电子控制单元的接口特征标准化，应用软件具备良好的可扩展性和可移植性，很大程度减小了开发周期。AUTOSAR 提倡“在标准上合作，在实现上竞争”的原则，其核心思想是“统一标准、分散实施、集中配置”。

2. AUTOSAR 模块构成

AUTOSAR Architecture 的分层式设计，用于支持完整的软件和硬件模块的独立性

(Independence),如图 3.3 所示,中间 RTE(Runtime Environment)作为虚拟功能总线 VFB(Virtual Functional Bus)的实现,隔离了上层的应用软件层(Application Layer)与下层的基础软件层(Basic Software),摆脱了以往 ECU 软件开发与验证时对硬件系统的依赖。

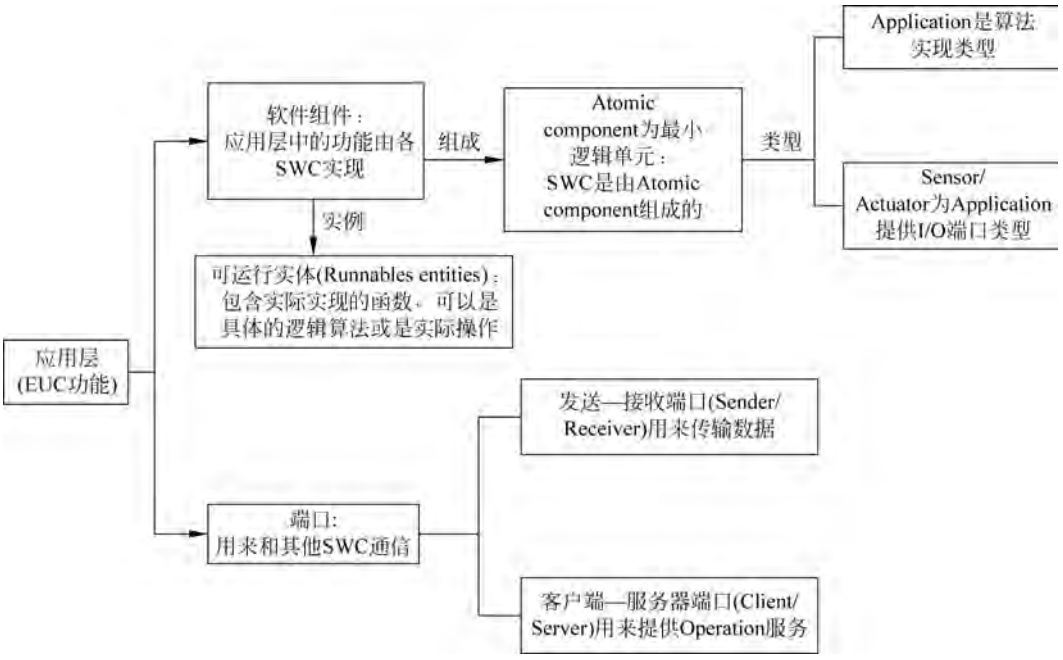


■图 3.3 AUTOSAR 体系架构分层标准

软硬件分离的分层设计,对厂商及供应商来讲,提高了系统的整合能力,特别是标准化交互接口以及软件组件模型的定义提高了各层的软件复用能力,从而减少了开发成本,提高了系统集成与产品推出的速度。

1) Application Layer(应用层)

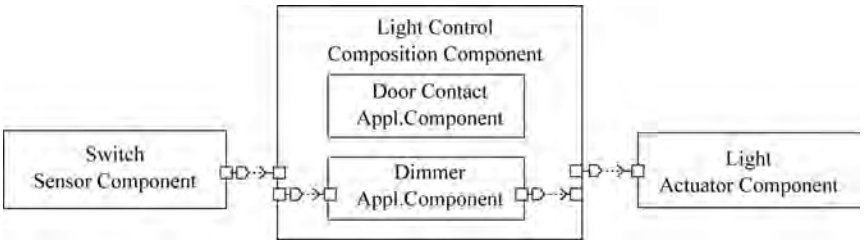
应用层中的功能由各软件组件(Software Component,SWC)实现,组件中封装了部分或者全部汽车电子功能,如图 3.4 所示。其中包括对其具体功能的实现以及对应描述,如控制大灯,空调等部件的运作,但与汽车硬件系统没有连接。



■图 3.4 应用层的组成

(1) 软件组件。

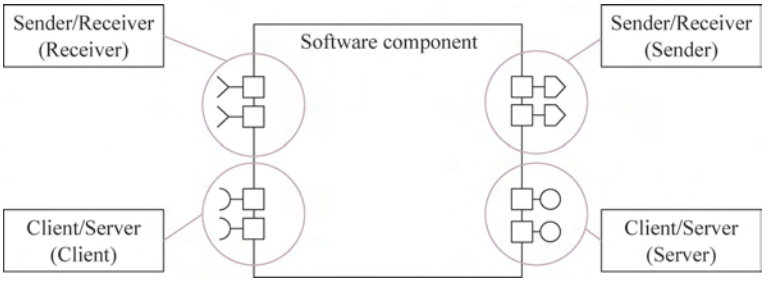
软件组件由最小逻辑单元(Atomic Component)组成。最小逻辑单元有 Application、Sensor/Actuator 两种类型。其中 Application 是算法实现类型,能在各 ECU 上自由映射;Sensor/Actuator 是为 Application 提供 I/O 端口类型,用于与 ECU 绑定,但不像 Application 那样能在各 ECU 上自由映射。数个 SWC 的逻辑集合组合成 Composition。图 3.5 是 SWC 组成实例图。



■图 3.5 SWC 的组成

(2) 端口。

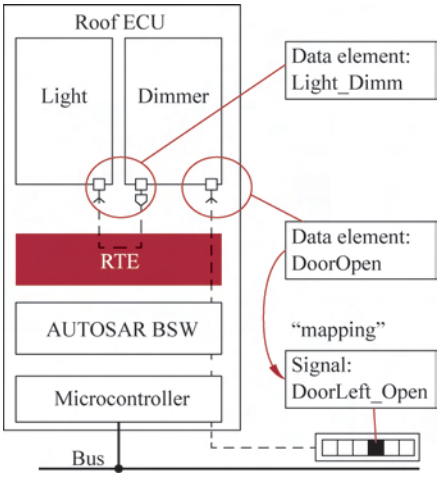
端口用来和其他 SWC 通信。通信内容分为 Data Elements 与 Operations。其中,Data Elements 用发送端—接收端(Sender/Receiver)通信方式,Operations 用客户端—服务器端(Client/Server)通信方式,如图 3.6 所示。



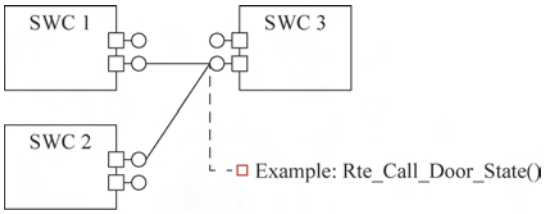
■图 3.6 通信方式

发送端—接收端(Sender/Receiver)用来传输数据,具有一个通信端口可以包含多种数据类型的特点。但如果一个数据类型要通过总线传输,那么它必须与一个信号对应起来,数据类型既可以是简单的数据类型(integer、float),也可以是复杂类型(array、record)。通信方式是 1∶n 或 n∶1,如图 3.7 所示。

客户端—服务器端口(Client/Serverr)用来提供 Operation 服务,具有一个客户端—服务器端口可以包含多种 Operation 和同步或是异步通信的特点,一个客户端—服务器端口可以包含多种 Operations 操作,Operations 操作也可被单独调用。通信方式是 1∶n 或 n∶1,如图 3.8 所示。



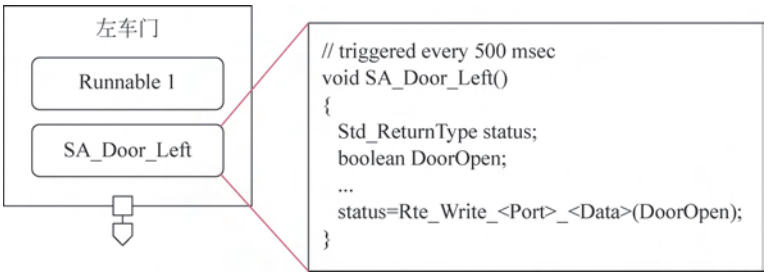
■图 3.7 发送端—接收端特征图



■图 3.8 客户端—服务器端口

(3) 可运行实体(Runnable Entities)。

可运行实体(Runnable Entities),简称 Runnables。可运行实体包含实际实现的函数,可以是具体的逻辑算法或是实际操作。可运行实体由 RTE 周期性或是事件触发调用,如接收到数据,如图 3.9 所示。



■图 3.9 可运行实体

2) RunTime Environment(中间件)

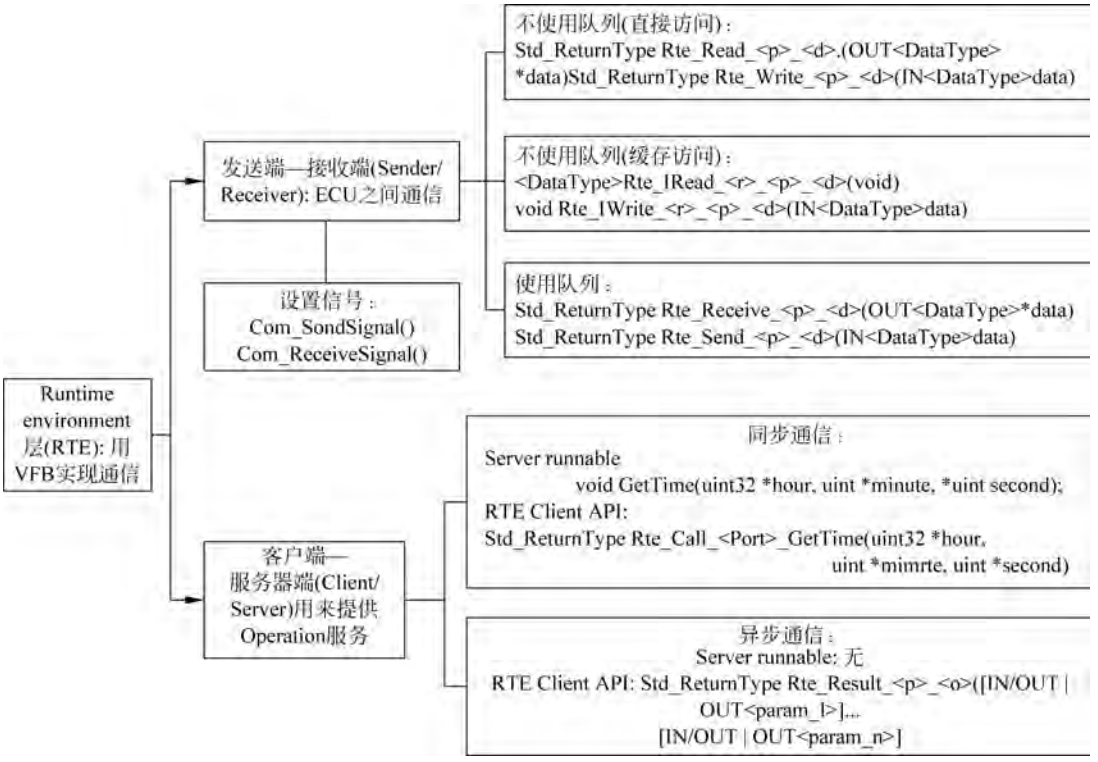
中间件(RTE)部分给应用层提供了通信手段,这里的通信是一种广义的通信,可以理解成接口。应用层与其他软件的信息交互有两种,第一种是应用层中不同模块之间的信息交互,第二种是应用层模块同基础软件之间的信息交互,其特征如图 3.10 所示。而 RTE 就是这些交互使用的接口的集散地,它汇总了所有需要和软件体外部交互的接口。从某种意义上来看,设计符合 AUTOSAR 规范的系统其实就是设计 RTE。

SWC 之间的通信是通过调用 RTE API 函数而非直接实现的,均由 RTE 管理和控制。每个 API 遵循统一的命名规则且仅与软件组件自身的描述有关。具体通信实现取决于系统设计和配置,由工具供应商提供的 RTE Generator 自动生成。

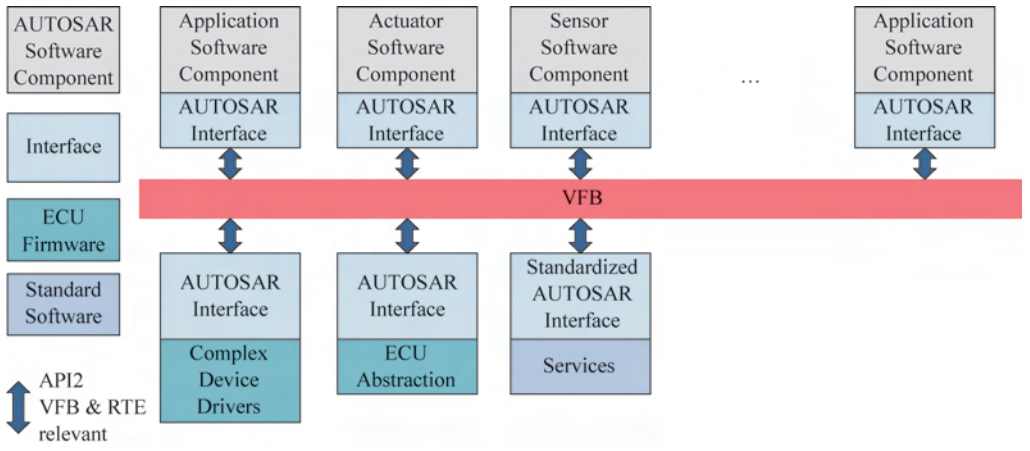
在设计开发阶段,软件组件通信层面引入了一个新的概念,虚拟功能总线(Virtual Functional Bus,VFB),如图 3.11 所示。它是对 AUTOSAR 所有通信机制的抽象。利用 VFB,开发工程师将软件组件的通信细节抽象,只需要通过 AUTOSAR 所定义的接口进行描述,即能够实现软件组件与其他组件以及硬件之间的通信,甚至是 ECU 内部或者是与其他 ECU 之间的数据传输。

从图 3.11 中可以看到,有 3 种接口描述,可以先从定义的角度来看这 3 种接口有什么不同。

(1) Standardized Interface(标准接口):标准接口是在 AUTOSAR 规范中被标准化的接口,但并未使用 AUTOSAR 接口技术,标准接口通常被用于某个 ECU 内部的软件模块之间的通信,不能用于网络通信。



■图 3.10 RTE 的特征图



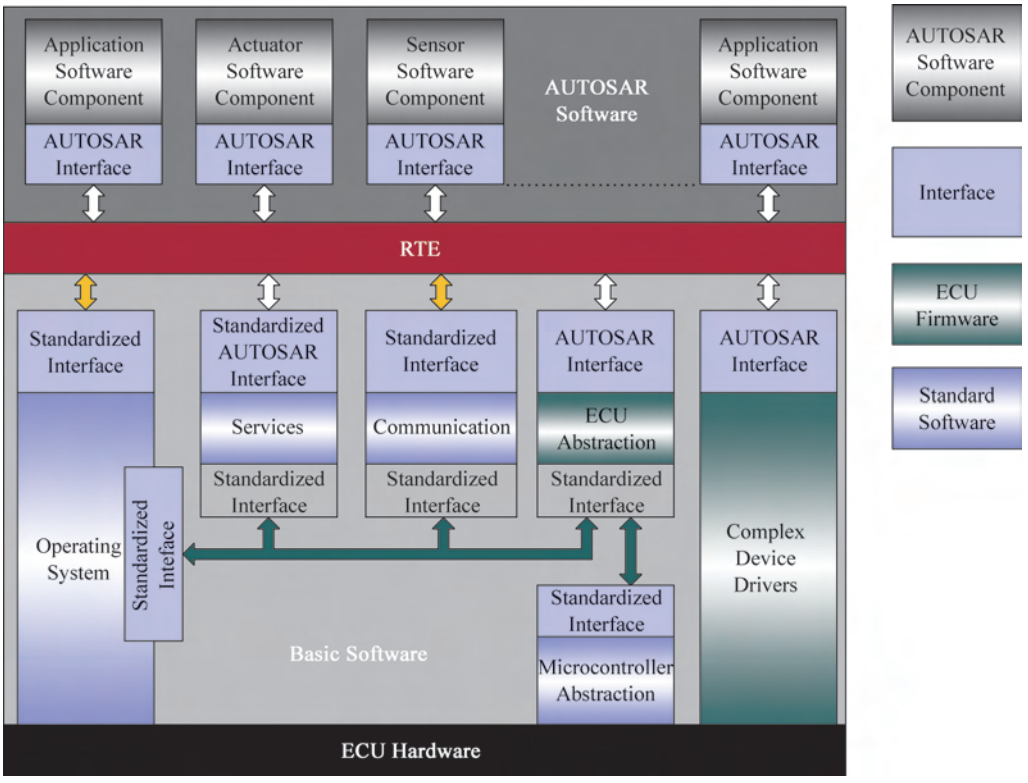
■图 3.11 虚拟功能总线

(2) Standardized AUTOSAR Interface(标准 AUTOSAR 接口)：标准 AUTOSAR 接口是在 AUTOSAR 规范中使用 AUTOSAR 接口技术标准化的接口，这种接口的语法和语义都被规定好了，通常在 AUTOSAR 服务中使用，是基础软件服务提供给应用程序的。

(3) AUTOSAR Interface(AUTOSAR 接口)：AUTOSAR 接口定义了软件模块和 BSW 模块(仅仅是 I/O 抽象和复杂驱动)之间交互的方式，AUTOSAR 接口以 Port 的形式出现，将 ECU 内部的通信和网络通信使用的接口进行了统一。

从以上定义中可以看出，不同的接口使用的场景不同，及不同的模块交互会使用不同的

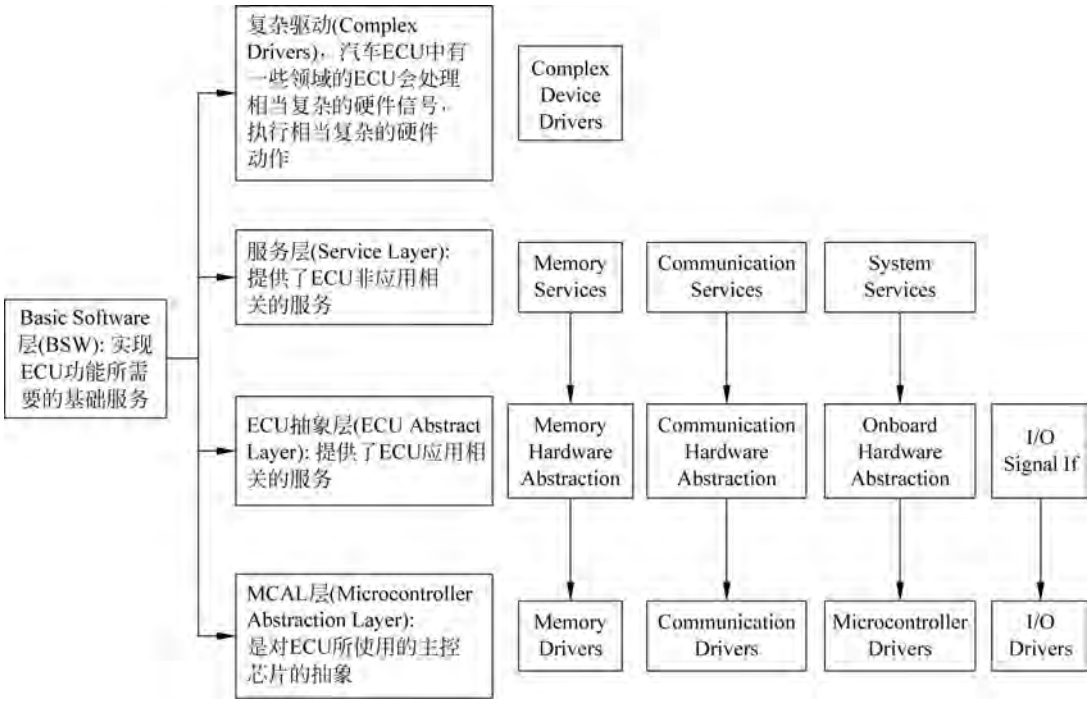
接口。除了将接口归类以外，这种定义的意义合在？从实际使用的角度来看，第一和第二类接口都是语法语义标准化的接口，即接口函数的数量、函数的名字、函数参数名字及数量、函数的功能、函数的返回值都已经在标准里边定义好了。不同公司的软件在实现这些接口的时候虽然内容算法不同，但是它们的长相和功能是一致的，接口定义在 AUTOSAR 规范文档里是可以查到的。第三类接口，AUTOSAR 仅仅规定了简单的命名规则，这类接口和应用高度相关，例如 BCU 控制大灯打开的接口可以是 Rte_Call_RPort_BeamLight_SetDigOut，也可以是 Rte_Call_RPort_HeaderLight_Output，车企可以自己定义。如仪表想要从 CAN 总线上获得车速，该接口可以定义为 Rte_IRead_RE_Test_RPort_Speed_uint8，也可以定义为 Rte_IRead_Test_RE_RPort_Spd_uint8，这些接口必须通过 RTE 交互，如图 3.12 所示。



■图 3.12 接口交互示意图

3) Basic Software(基础软件)

现代汽车中有多种 ECU，各自具有不同功能，但实现这些功能所需要的基础服务是可以抽象出来的，例如 I/O 操作、AD 操作、诊断、CAN 通信、操作系统等；差别是，不同的 ECU 功能所操作的 I/O、AD 代表不同的含义，所接收和发送的 CAN 消息具有不同的内容，操作系统调度的任务周期优先级不同。这些可以被抽象出来的基础服务被称为基础软件。根据不同的功能对基础软件可继续细分成 4 部分，分别为服务层(Service Layer)、ECU 抽象层(ECU Abstract Layer)、复杂驱动(Complex Drivers)和 MCAL 层(Microcontroller Abstraction Layer)，4 部分之间的互相依赖程度不尽相同，如图 3.13 所示。



■图 3.13 BSW 的结构组成

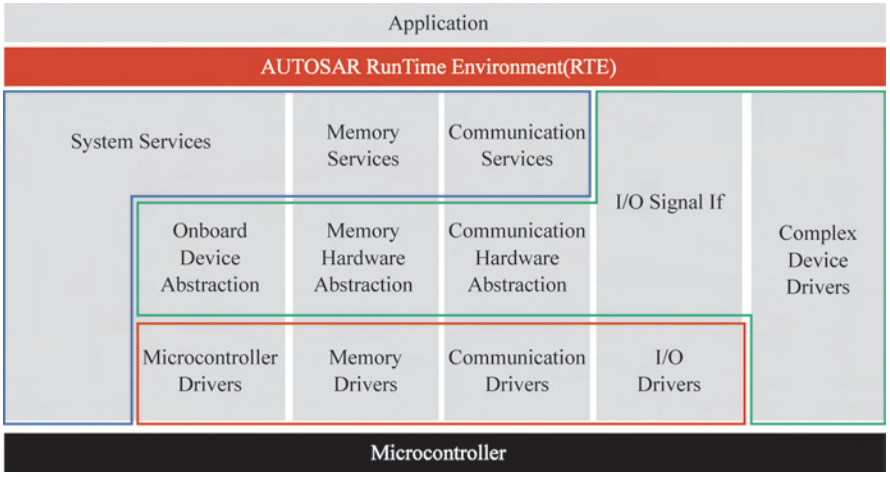
服务层(Service Layer),这一层基础软件提供了汽车 ECU 非应用相关的服务,包括操作系统(OS)、网络通信、内存管理(NVRAM)、诊断(UDS,故障管理等)、ECU 状态管理模块等,它们对 ECU 的应用层功能提供辅助支持。该层软件在不同领域的 ECU 中也非常相似,例如不同的 ECU 中的 OS 的任务周期和优先级不同,不同的 ECU 中的 NVRAM 的分区不同,存储的内容不同。

ECU 抽象层(ECU Abstract Layer),这一层软件提供了 ECU 应用相关的服务,它是对一个 ECU 的抽象,包括 ECU 的所有输入输出,例如数模转换(AD)、PWM 等。该层软件直接实现了 ECU 的应用层功能,可以读取传感器状态,可以控制执行器输出。不同领域的 ECU 会有很大的不同。

MCAL 层(Microcontroller Abstraction Layer),这一层软件是对 ECU 所使用的主控芯片的抽象,它跟芯片的实现紧密相关,是 ECU 软件的最底层部分,直接和主控芯片及外设芯片进行交互。其作用是将芯片提供的功能抽象成接口,再将这些接口提供给服务层或 ECU 抽象层使用。

复杂驱动(Complex Drivers),现代汽车中有一些领域的 ECU 会处理复杂的硬件信号,执行复杂的硬件动作,例如发动机控制、ABS 等。这些功能相关的软件很难抽象出来适用于所有类型 ECU,其与 ECU 应用及 ECU 所使用的硬件紧密相关,属于在 AUTOSAR 构架中无法移植于不同 ECU 的部分。图 3.14 是 BSW 层中各个子模块说明。

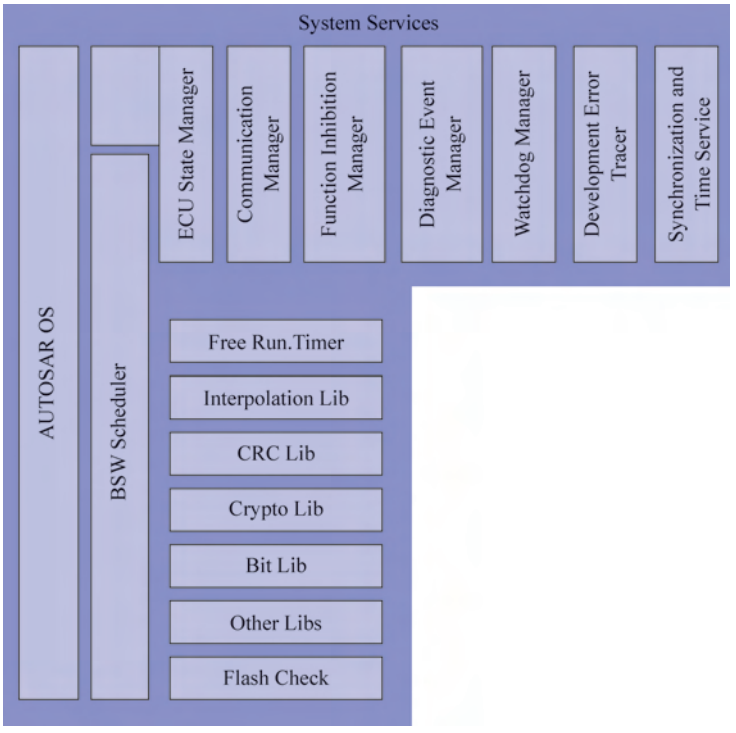
微控制器抽象层是基础软件中的最底层。它包含各种驱动模块,用来对 μC 内部设备和映射了 μC 外部设备的内存进行访问。



■图 3.14 BSW 子模块示意图

4) System Services(系统服务)

系统服务是一组可以由所有层模块调用的模块和功能,例如实时操作系统、错误管理器和库功能,为应用和基本软件模块提供基本服务。具体包括 AUTOSAR OS、BSW 调度器和模式管理器 3 个部分,如图 3.15 所示,下面分别介绍。



■图 3.15 系统服务的结构

(1) AUTOSAR OS 介绍。

AUTOSAR OS 为实时应用提供了所有基本服务,即中断处理、调度、系统时间和时钟

同步、本地消息处理，以及错误检测机制。所有服务都隐藏在良好定义的 API 之后。应用与 OS 和通信层的连接只通过 API。其基本特征包括：静态配置、能推断实时系统性能和提供基于优先级的调度策略等。

在嵌入式汽车 ECU 中的实时操作系统构成软件动态行为的基础。它管理任务和事件的调度和不同任务间的数据流，并且提供监控和错误处理功能。但是，在汽车系统中，对操作系统的需求集中在特定领域，所使用的操作系统必须高效运行并且所占存储空间小。在多媒体和远程信息处理应用中，操作系统提供的特征集以及可用计算资源有很大不同。在纯粹的任务管理之上，操作系统中还包含了复杂的数据处理（例如，流、快速文件系统等）和存储管理甚至图形用户接口。

汽车操作系统的典型领域涵盖了调度和同步的核心特征。在 AUTOSAR 中，上面讨论的附加特征在 OS 的范围之外，其他 WP4.2.2.1 工作包（例如 SPAL）涵盖了这些特征。在 AUTOSAR 的体系结构约束之下不可能把其他 OS（例如，QNX、VxWorks 和 Windows CE 等）的特征集合集成到整体的 OS/通信/驱动结构中。因此，AUTOSAR OS 只考虑核心特征。

（2）BSW 调度器。

BSW 调度器是系统服务的一部分，它向所有层的所有模块提供服务。但是，与其他 BSW 模块不同，BSW 调度器提供了集成的概念和服务。BSW 调度器可提供方法把 BSW 模块的实现嵌入 AUTOSAR OS 中，并应用 BSW 模块的数据一致性机制。集成者的任务是应用所给的方法（AUTOSAR OS 提供的），在特定项目环境中以良好定义和有效的方式把 BSW 模块装配起来。这也意味着 BSW 调度器只是使用 AUTOSAR OS，它与 AUTOSAR OS 调度器并不冲突。

（3）模式管理。

模式管理包括 3 个基本软件模块：①ECU 状态管理器，控制 AUTOSAR BSW 模块的启动阶段，包括 OS 的启动；②通信管理器，负责网络资源管理；③看门狗管理器，基于应用软件的生存状态触发看门狗。

3.2.2 Apollo 软件架构

1. Apollo 概述

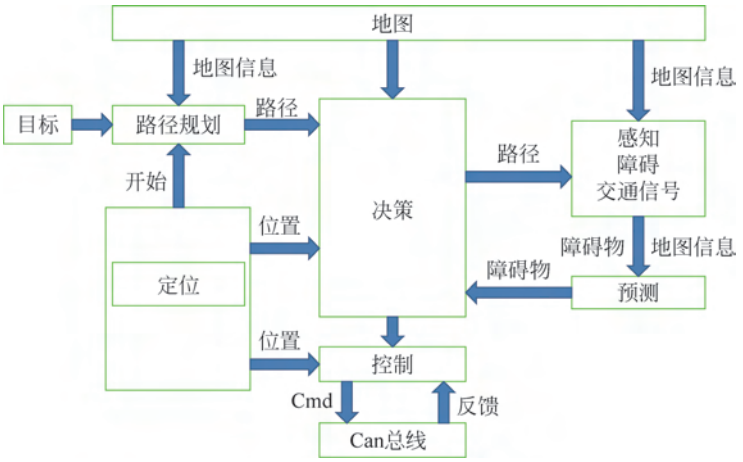
Apollo 是百度面向汽车行业及自动驾驶领域合作伙伴发布的，一个开放、完整、安全的自动驾驶平台。通过这个平台，开发者可整合车辆和硬件系统，搭建完整的自动驾驶系统。Apollo 平台提供的软硬件和服务，包括车辆平台、硬件平台、软件平台、云端数据服务等 4 大部分。

Apollo 目前已经开放 Apollo 1.0 封闭场景循迹自动驾驶、Apollo 1.5 昼夜定车道自动驾驶、Apollo 2.0 简单城市道路自动驾驶、Apollo 2.5 限定区域视觉高速自动驾驶、Apollo 3.0 量产园区自动驾驶和 Apollo 3.5 带来的 Apollo Enterprise 5 个版本，完成平台 4 大模块全部开源，提供更多场景、更低成本、更高性能的能力支持，为自动驾驶车辆量产提供软件、硬件、安全、多模人机交互方面的全面平台服务支持。

2. Apollo 软件构架

Apollo 软件架构由以下几个模块组成：校准模块车辆 CAN 总线控制模块、控制模块、

决策模块、可视化模块、驱动模块、人机交互模块、端到端强化学习、定位模块、高精地图模块、监控模块、感知模块、预测模块以及通用监控与可视化工具。自动驾驶系统先通过起点终点规划出全局路径(Routing)；然后在行驶过程中感知(Perception)当前环境(识别车辆行人路况标志等)，并预测下一步发展；然后将采集到信息传入规划模块(Planning)，规划车辆行驶轨迹；控制模块(Control)将轨迹数据转换成对车辆的控制信号，通过汽车交互模块(Canbus)控制汽车，如图 3.16 所示。



■图 3.16 Apollo 平台各模块之间的关系

(1) 校准模块：使用前必须对传感器校准和标定，包括激光雷达与摄像头、毫米波雷达与摄像头等。校准是对齐激光雷达、摄像头以及毫米波雷达获得的信息。激光雷达可以获取详细的三维环境信息，但是不能获得颜色信息；摄像头可以获取颜色信息，但无法获得深度等三维信息；毫米波雷达不能获取颜色信息，但可以获得三维信息；将三者采集的信息对齐后，就可以同时了解实际环境中的三维信息和颜色信息。

(2) 车辆 CAN 总线控制模块：接收控制指令，同时给控制模块 Control 发送车身状态信息。CanBus 有两个数据接口。第一个数据接口是基于计时器的发布者，回调函数为 OnTimer。如果启用，此数据接口会定期发布底盘信息。第二个数据接口是一个基于事件的发布者，回调函数为 OnControlCommand，当 CanBus 模块接收到控制命令时会触发该函数。

(3) 控制模块：基于决策规划的输出路径及车身状态，使用不同的控制算法来输出控制命令，如转向、制动、加速等。有 3 个主要的数据接口：OnPad、OnMonitor 和 OnTimer。OnPad 和 OnMonitor 是仿真和 HMI 的交互接口。主要数据接口 OnTimer 会定期产生实际的控制命令。

(4) 决策模块：在接收全局路径后，根据从感知模块得到的环境信息(其他车辆、行人等障碍物信息，道路上交通标志、红绿灯等交通规则信息)，以及本车当前的行驶路径等状态信息，做出具体的行为决策(如变道超车、跟车行驶、让行、停车、进出站等)。

(5) 可视化模块：查看规划的轨迹及实时的转向、制动和节气门信息。

(6) 驱动模块：GNSS 设备驱动，包括 NovAtel、Applanix、u-blox、激光雷达、Velodyne 驱动，用来读取传感器内容并输出对应的消息。

(7) 人机交互模块：可视化自动驾驶模块的输出，例如，规划轨迹、汽车定位、底盘状态

等。为用户提供人机交互界面，以查看硬件状态，打开或关闭模块，以及启动自动驾驶汽车。

(8) 端到端强化学习：所谓端到端是指，由传感器的输入信息，直接决定车的行为，例如节气门、制动、方向等。这是机器学习算法直接学习人类司机的驾驶行为。学习数据主要来源于传感器的原始数据，包括图像、激光雷达、毫米波雷达等。端到端输入以图像为主，输出车辆的控制决策指令，如方向盘角度、加速、制动等。通过深度神经网络连接输入输出两端，即神经网络直接发送车辆控制指令从而对车辆进行横向和纵向控制，此过程没有人工参与。横向控制，主要是指通过方向盘控制车身横向移动，即方向盘角度。纵向控制，是指通过节气门和制动控制车身纵向的移动，即加速、制动等。横向模型的输出不是方向盘角度，而是道路行驶的曲率。

(9) 定位模块：聚合各种数据以定位自动驾驶车辆，有两种类型的定位模式，OnTimer和多传感器融合。第一种基于 RTK 的定位方法，通过计时器的回调函数“OnTimer”实现；另一种是多传感器融合(MSF)方法，其中注册了一些事件触发的回调函数。

(10) 高精地图模块：输出结构化地图信息，如车道线、十字路口等。其显著特性是表征路面特征的准确性。通常，传统地图只需要做到米量级的精度即可实现基于 GPS 的导航，但高精地图需要至少 100 倍以上的精度，即达到厘米级的精度才能保证无人车的行驶安全。

(11) 监控模块：包括硬件在内的，车辆中所有模块的监控系统。监控模块从其他模块接收数据并传递给 HMI，以便驾驶员查看并确保所有模块都正常工作。如果模块或硬件发生故障，监控会向 Guardian 发送警报，然后决定需要采取哪些操作来防止系统崩溃。

(12) 感知模块：感知依赖 LiDAR 点云数据和摄像头原始数据。除了传感器数据输入之外，交通灯检测也需要依赖定位以及高精地图。由于实时 ad-hoc 交通灯检测在计算上是不可行的，因此需要依赖定位确定何时何地开始通过摄像头捕获的图像检测交通灯。在感知模块中，Apollo 平台有以下几个特点：①CIPV 检测/尾随，可实现远程精确度。摄像头安装有高低两种不同的安装方式。②异步传感器融合，因为不同传感器的帧速率差异，毫米波雷达为 10ms，摄像头为 33ms，LiDAR 为 100ms，所以异步融合 LiDAR、毫米波雷达和摄像头数据，并获取所有信息得到数据点的功能非常重要。③在线姿态估计，自动驾驶汽车在出现颠簸或斜坡时需确定与估算角度变化，以确保传感器随汽车移动且角度或姿态相应地变化。④超声波传感器，作为安全保障传感器，与 Guardian 一起用于自动紧急制动和停车。

(13) 预测模块：负责预测所有感知到的障碍物的未来运动轨迹，输出预测消息封装了感知信息。预测定位和感知障碍物消息时，当接收到定位更新后，预测模块更新其内部状态。当感知模块发布感知到障碍物消息时，触发预测模块实际执行。

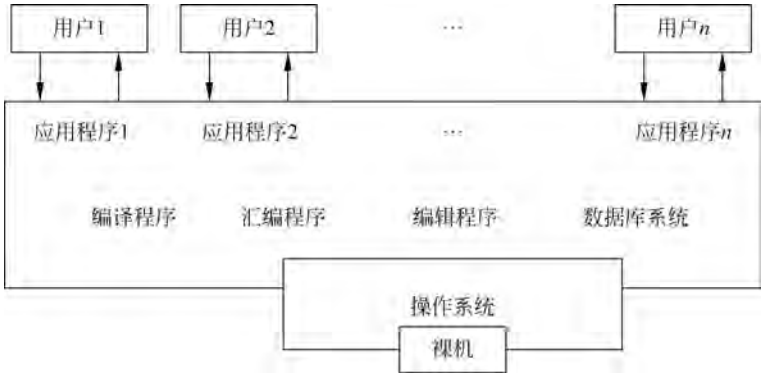
(14) 通用监控与可视化工具：包括一些可视化工具，bag 包录制及回放脚本。

3.3 自动驾驶汽车操作系统

在软件平台中，操作系统主要管理系统中的各种软、硬件资源，控制用户和应用程序的工作流程。操作系统是架构在硬件之上的第一层软件，是系统软件和应用程序运行的基础。软件平台和操作系统的关系如图 3.17 所示。

自动驾驶汽车的操作系统负责管理车辆对四周物体的识别、车辆定位及路径规划等功

能，它是实现无人驾驶的关键。由于自动驾驶汽车具有强安全关联属性，若操作系统功能欠佳，其代价不仅是工作效率低下，而关乎生命安全，所以自动驾驶汽车的操作系统在监控支配汽车时的反应需要精确到微秒级，能够实时感知周围环境并规划出针对性的解决方案。车载操作系统分为实时操作系统和非实时操作系统，下面将详细介绍两个车载操作系统。



■图 3.17 软件平台与操作系统关系图

3.3.1 车载实时操作系统

车载实时操作系统将分为 4 个部分展开介绍，其中包括实时操作系统(RTOS)的定义，实时操作系统的特征，实时操作系统的相关概念和典型车载实时操作系统 μC/OS-Ⅱ。

1. 实时操作系统定义

实时操作系统是确保在规定时间内完成指定功能的操作系统。例如，可以为自动驾驶汽车的路面状况实时判断设计一个操作系统。

实时操作系统最显著的特征是实时性，即当外部数据或者请求到来之时，在极其短暂(微秒级)的时间内做出中断响应，并且交由 CPU 进行处理。另外，高可靠性也是特征之一，如果有特殊情况发生，它能某个时间范围内得到处理。

实时操作系统可以被划分为硬实时(hard realtime)和软实时(soft realtime)操作系统，划分标准为硬性截止时间的不同。硬实时系统是指要在最坏情况(负载最重)下确保在服务时间内完成响应，即对于事件的截至响应期限必须在规定时间内满足(一次也不能超时)。软实时系统是指在规定时间以内，尽量保证处理完成相关任务和数据，当出现了超时情况，也属于可以接受的范围，并不会造成严重的后果(允许有限次数内超时)。

对于在实际应用过程中所出现的相同情况，硬实时系统和软实时系统会采用完全不同的应对策略。如果因为延期导致错过了任务的执行截止时间，硬实时系统会选择直接结束当前工作，然后关闭系统；而软实时系统仅仅放弃执行当前任务，可能会有短暂的暂停，然后转为执行队列中的下一个就绪任务。现在市场上所存在的系统一部分是为特定的实时应用所设计，另一部分是通用的软实时操作系统，如 Windows NT 或 IBM 的 OS/390。

2. 实时操作系统的特征

1) 高精度计时系统

计时精度是影响实时性的一个重要因素。在实时应用系统中，经常需要精确实时地操

作某个设备或执行某个任务，或精确地计算一个时间函数。这些不仅依赖一些硬件提供的时钟精度，也依赖实时操作系统实现的高精度计时功能。

2）多级中断机制

一个实时操作系统通常需要处理多种外部信息或事件，但处理的紧迫程度有轻重缓急之分。有的必须立即作出反应，有的则可以延后处理。因此，需要建立多级中断嵌套处理机制，以确保对紧迫程度较高的实时事件及时进行响应和处理。

3）实时调度机制

实时操作系统不仅要及时响应事件中断，同时也要及时调度运行实时任务。但是处理机调度并不能随心所欲地进行，因为涉及两个进程之间的切换，只能在确保“安全切换”的时间点上进行。实时调度机制包括两个方面：一是在调度策略和算法上保证优先调度实时任务；二是建立更多“安全切换”时间点，保证及时调度实时任务。

3. 实时操作系统的相关概念

1）基本概念

代码临界段：在一个时间段内，只允许一个线程或进程进行独占式访问的代码段。其他所有试图访问该代码段的进程都必须进行等待。

资源：进程所占用的任何实体。

共享资源：可以被多个进程共享的一次具体活动，以进程或者线程的方式存在，拥有自己的地址空间（包括文本、数据和堆栈共同使用的实体）。通过一系列操作达到某一目的，例如使用打印机打印出一串字符。拥有 4 种常见状态：休眠态、就绪态、运行态、挂起态。

任务切换：当系统中存在两个或两个以上的任务时，处于就绪态任务需要抢占运行态任务，或者运行态任务执行完毕，需要让出 CPU 控制权而进行的切换操作；当前占据 CPU 使用权的任务存入栈区，将下一个即将开始的任务装入 CPU 寄存器，开始运行。

内核：操作系统的核心，是硬件层和软件层的交互媒介，提供操作系统的基本功能。负责对任务的管理、CPU 调度、设备驱动、内存管理等，可以分为抢占式和非抢占式。

调度：当多个进程向同一资源发出请求时，由于访问互斥性，必须按照一定优先次序对唯一性资源进行分配。

2）关于优先级的问题

任务优先级根据运行过程中是否恒定，分为静态优先级和动态优先级。在实际应用过程中，经常会遇到优先级翻转的问题，导致优先级较低的进程长时间占据共享资源，阻塞了高优先级任务的运行。目前，有“优先级天花板”和“优先级继承”两种办法可以解决这个问题。第一种方法是在所有申请同一共享资源的任务中选出一个最高优先级，把这个优先级赋给每一个申请该资源的任务，即不让某一资源的优先级成为影响高优先级的任务运行的关键因素；第二种方法是当出现高优先级任务被低优先级任务占用临界区资源时，在一定时间内将低优先级任务提高到与高优先级任务相同，缺点是每一项新任务到来时，都需要进行判断。

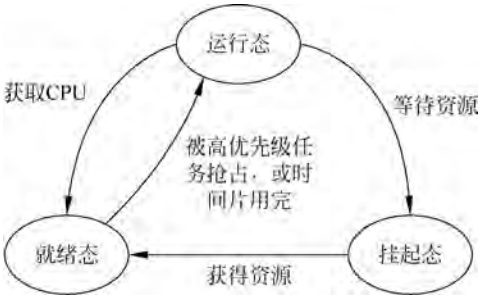
3）互斥

共享内存的意义在于可以让进程之间的通信变得方便、迅速，但是当一个进程对该区域进行读写的时候，为了防止脏读脏写，必须保证访问的互斥性，即其余请求访问该内存区域的进程必须等待，直到此内存块被释放。实现互斥访问一般包括软件和硬件两种方法，软件

方法比较著名的有 DEKKER 算法和 PETERSON 算法(更好);硬件方法主要是通过特殊指令来达到保护临界区的目的,包括忙等待、自旋锁、开关中断指令、测试并加锁指令、交换指令等。每一种方案都是有利有弊,以开关中断指令举例,主要优点是简单高效,但是代价高,不利于发挥 CPU 的并发能力,并且只适用于单核处理器,仅仅适用于操作系统本身,而无法适用于应用程序。

4) 任务切换时间

任务切换的时间是衡量一个实时操作系统实时性能的重要指标之一,其取决于 CPU 需要等待入栈的寄存器个数。CPU 寄存器个数越多,则切换时间越长。任务切换时的状态,如图 3.18 所示。



■图 3.18 任务切换状态图

5) 中断响应时间(可屏蔽中断)

中断响应时间是另一个衡量实时操作系统实时性能的重要指标,其主要由关中断的最长时间、保护 CPU 内部寄存器的时间、进入中断服务函数的执行时间、开始执行中断服务程序(ISR)的第一条指令时间构成。

4. μC/OS-Ⅱ概述

μC/OS-Ⅱ由 Micrium 公司提供,是一个可移植、可固化、可裁剪的,占先式多任务实时内核,它适用于多种微处理器、微控制器和数字处理芯片(已经移植到超过 100 种以上的微处理器应用中)。同时,该系统源代码开放、整洁、一致、注释详尽,适合系统开发。μC/OS-Ⅱ已经通过联邦航空局(FAA)商用航行器认证,符合航空无线电技术委员会(RTCA)DO-178B 标准。μC/OS-Ⅱ可以大致分成核心、任务处理、时间处理、任务同步与通信,CPU 的移植等 5 个部分。

(1) 核心部分(OSCore.c)是操作系统的处理核心,包括操作系统初始化、操作系统运行、中断进出的前导、时钟节拍、任务调度、事件处理等多部分。能够维持系统基本工作的部分都在这里。

(2) 任务处理部分(OSTask.c),其内容都是与任务的操作密切相关。包括任务的建立、删除、挂起、恢复等。因为 μC/OS-Ⅱ是以任务为基本单位进行调度,所以这部分内容也相当重要。

(3) 时钟部分(OSTime.c),μC/OS-Ⅱ中的最小时钟单位是 timetick(时钟节拍)。任务延时等操作是在这里完成的。

(4) 任务同步和通信部分,为事件处理部分,包括信号量、邮箱、邮箱队列、事件标志等

部分；主要用于任务间的互相联系和对临界资源的访问。

(5) 与 CPU 的接口部分，是指 μC/OS-Ⅱ针对所使用的 CPU 的移植部分。由于 μC/OS-Ⅱ是一个通用性的操作系统，所以对于关键问题上的实现，还是需要根据具体 CPU 的内容和要求作相应的移植。这部分内容由于牵涉 SP 等系统指针，所以通常用汇编语言编写，主要包括中断级任务切换的底层实现、任务级任务切换的底层实现、时钟节拍的产生和处理、中断的相关处理部分等内容。

3.3.2 车载准实时操作系统

1. 准实时操作系统定义

准实时操作系统是使一台计算机采用时间片轮转的方式同时为几个、几十个甚至几百个用户服务的一种操作系统。为把计算机与许多终端用户连接起来，准实时操作系统将系统处理机时间与内存空间按一定的时间间隔，轮流地切换给各终端用户的程序使用。由于时间间隔很短，每个用户的感觉就像他在独占计算机一样。准实时操作系统的特点是可有效增加资源的使用率。

准实时操作系统典型的例子就是 UNIX 和 Linux 的操作系统。其可以同时连接多个终端并且每隔一段时间重新扫描进程，重新分配进程的优先级，动态分配系统资源。

2. 准实时操作系统的特征

准实时操作系统有以下几个特征。

(1) 交互性(同时性)：用户与系统进行人机对话。用户在终端上可以直接输入、调试和运行自己的程序，在本机上修改程序中的错误，直接获得结果。

(2) 多路性(多用户同时性)：多用户同时在各自终端上使用同一 CPU 和其他资源，充分发挥系统的效率。

(3) 独立性：用户可彼此独立操作，互不干扰，互不混淆。

(4) 及时性：用户在短时间内可得到系统的及时回答。

影响响应时间的因素：主要有终端数目多少、时间片的大小、信息交换量、信息交换速度等。

3. Linux 系统结构

在图 3.19 中，可以看出 Linux 系统是一个分层的体系结构，位于硬件层之上，由用户空间和内核空间组成，其中高位的物理内存由内核空间占用，这部分内存只限运行在内核态的进程访问，主要划分为 ZONE_DMA、ZONE_NORMAL 和 ZONE_HIGHMEM 3 部分。低位的物理内存由用户空间使用，进程对用户空间的访问是互相隔离的，某一时刻占据 CPU 的进程，拥有整个虚拟内存空间。

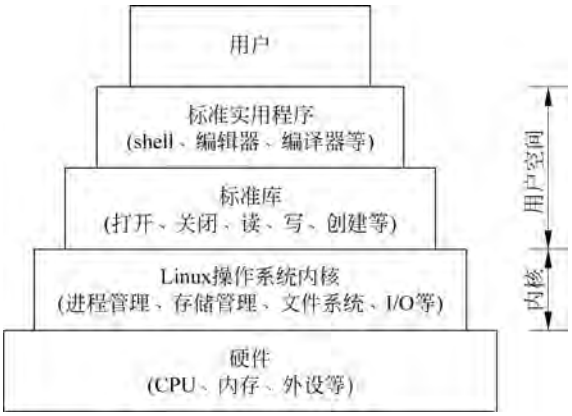
所有实时应用，都需要在极短的反应时间内，满足系统的任务处理需求。目前版本的 Linux 系统由于以下的几个自带特性，是无法满足硬实时操作系统要求的。

(1) 时间粒度太大(毫秒级)。

(2) 时间片轮转调度策略。

(3) 虚拟内存管理。

(4) 非抢占式内核。



■图 3.19 Linux 系统结构图

(5) 临界区中断屏蔽。

Linux 进程的运行状态一般分为阻塞态、就绪态和运行态 3 种。当一个进程所有资源都具备,只差 CPU 控制权的时候,会根据优先级被插入就绪队列合适位置,进入就绪态。当就绪队列的进程获得 CPU 之后,进入运行态,直至时间片用完,进入新的就绪队列;或者 I/O 设备等硬件资源被剥夺,则进入阻塞态,直到重新获取硬件资源,再由系统移至就绪队列末尾继续等待。在 Linux 系统中,一旦进程进入运行状态,即便就绪队列中有更高优先级任务到达,也无法立即对运行中的低优先级进程进行抢占。除非当前运行的低优先级进程自动放弃 CPU,或者等待硬件资源被剥夺而造成阻塞,否则将一直处于运行态。这个时间间隔甚至可能达到毫秒级别,这对于实时性要求很高的系统是完全无法容忍的。此外,当 Linux 系统在对临界区资源进行操作时,中断标志位是处于屏蔽状态,即无法通过中断请求,而立刻插入执行高实时性任务。等待的时间长度完全取决于系统调用所耗费的时间,这段时间内系统无法处理外部的中断请求。实时性操作系统要求,对于低优先级的当前运行任务,只要就绪队列中存在优先级比它高的任务,就可以立刻剥夺它的运行权利。

Linux 采用时间片轮转调度策略,每个进程都被分配相同长度的时间片,从获取 CPU 运行权开始计时,当期时间片用完之时,不管进程是否运行完毕,都必须交出 CPU 控制权。此调度策略的优点是每一个等待队列中的进程在一定时间内都有机会获得执行权力,防止出现某一进程因等待时间过长而挂死的情况;缺点是对于优先级高的长进程,可能需要多次调度执行,才能完成任务,降低了系统的平均执行效率。Linux 采取将任务集中进行分配处理的方式,这会导致任务超时未被处理的情况出现。在用户态下运行的进程,随着时间片用完或者缺少相关硬件资源造成的等待,会被内核剥夺处理器的运行权;而一旦将运行状态切换到内核态,则可以不受限制地一直运行下去,直到自己主动提出放弃请求,或者切回用户态。当系统出现不可预知情况时,优先级是可以通过系统管理者进行手动修改的。这样做有利于增强复杂环境的适应性,但是也会对程序调度的既定性造成影响。

当出现随机存储器(RAM)不够用的时候,Linux 采用虚拟内存技术,通过将内存映射到扩展的虚拟存储器(外部磁盘),达到加大内存容量的效果。首先,这需要建立一个额外的

数据结构来管理这部分虚拟内存；其次，缺页所造成的页面换进换出和磁盘 I/O 操作需要一笔不小的时间开支，并且这段时间支出是无法预知的。如果这段时间里，有高优先级任务到来，系统是无法提供及时响应的。这也是 Linux 无法作为实时操作系统的一个重要原因。但是，Linux 提供了丰富的硬件支持、免费开源的工具库，这是其他实时操作系统无法比拟的。因此，为了利用 Linux 的这些资源优势，当前主要有两种处理方案。

第一种方案是直接改动内核的源代码，通过对周期模式、数据结构、调度方式、中断屏蔽等进行改动，达到提高实时性的目的。这种方式改动后的实时性很好，原来的内核模块依旧可以使用，在其上编写代码和普通 Linux 基本相同，可以系统调用；缺点是工作量很大，并且后期会存在系统稳定性问题。这种方法是针对 Linux 内核的进程调度算法做部分修改，减少因进程的不可抢占所导致的时间耗费。采用这种方法的目的是降低系统的“中断延迟”和任务的“调度延迟”，提升内核的实时处理能力。此法可以保证在一个较短时间内完成对高优先级任务的响应，但是却无法保证具体的响应时间，因此提供的是一个软实时系统。

第二种是采用双内核的方式，在 Linux 内核外部进行扩展。目前，比较常见的做法是将 RTAI 或 xenomai 实时内核在原有 Linux 内核的基础上打补丁。这两款实时内核是当前比较成熟的硬实时内核，但面临的问题是项目的可维护性和二次开发难度，基本就是重写了一个内核系统，不具备大规模应用开发的基础。采用双内核后，普通进程使用 Linux 内核调度器进行调度，实时任务使用新内核调度器进行调度。这种双内核系统的两个内核互相配合，合理分工，当外部请求到来时，先由系统划分优先级，将优先级高的进程分配给实时内核进行处理，优先级低的进程分配给非实时内核进行处理。因此这种改造方法兼具了系统的稳定和工作量小两大优点。

这种方法通过在硬件层和软件层中间直接加入一个硬件请求管理层，专门负责截取和分发底层硬件请求，一般称为硬件资源抽象层(Hardware Abstraction Layer，HAL)。它可以提供一种兼容环境，允许多个操作系统或者一个操作系统中的多个进程共享资源，这些共享资源的系统或者进程被称作“域”。各个域之间彼此可能不知道对方的存在，但是它们可以通过 HAL 提供的中断管道互相通信。当外部中断、系统调用或者由于任务切换、进程创建等引起的系统事件发生时，HAL 负责依次告知各个域。

所有的外部硬件任务到来之时，都会对 HAL 发起中断请求。经过 HAL 的任务分级处理，每个任务会被分到实时任务序列或者非实时任务序列，根据实时性分别分配给两个内核。

采用独立实时调度管理机制与硬件抽象层的优势是对 Linux 内核部分的改动很少，并且可以兼容大部分的硬件驱动，便于应用程序的开发和移植。

3.4 本章小结

本章详细介绍了自动驾驶汽车软件平台的相关内容，由宏观的软件平台架构为起点，系统地总结了 AUTOSAR 系统以及典型 Apollo 软件平台的组成，接着对车载实时操作系统

和准实时操作系统进行了简要论述，分别介绍了各自的特点。随着汽车信息化、网络化、智能化快速发展，软件平台越来越体现出其在自动驾驶汽车开发中发挥的基础和核心作用。

参考文献

[1] 中国电子信息产业发展研究院，王鹏. 2013—2014年中国软件产业发展蓝皮书：中国软件产业发展蓝皮书[M]. 北京：人民出版社，2014.

[2] 赛迪网. 软件成智能汽车突破发展的关键[EB/OL]. (2016-09-29)[2019-3-10]. http://www.ccidnet.com/2016/0929/10190373.shtml。

[3] 韩健. 软件成智能汽车突破发展的关键[N]. 中国计算机报，2016-09-26(002).

[4] 黄铁雄. 开放式发动机管理系统体系结构及其虚拟原型技术研究[D]. 华中科技大学，2011.

[5] 陈海兰. 基于AUTOSAR规范的嵌入式实时操作系统设计与实现[D]. 复旦大学，2013.

[6] 高嵘昊. 面向汽车电子领域的嵌入式软件可靠技术的研究与开发[D]. 电子科技大学，2012.

[7] 徐志刚，张宇琴，王羽，等. 我国自动驾驶汽车行业发展现状及存在问题的探讨[J]. 汽车实用技术，2019，280(01)：21-29.

[8] 电子说. 汽车产业现况及自动驾驶的挑战[EB/OL]. (2018-05-22)[2019-3-10]. http://www.elecfans.com/d/680878.html.

[9] 彭威. SmartSAR RTE[D]. 浙江大学，2011.

[10] 戴筱妍. 车辆动力传动综合控制系统设计方法及关键技术研究[D]. 北京理工大学，2014.

[11] 扈杭. 从百度Apollo计划探讨无人驾驶技术的发展[J]. 数字通信世界，2017(09)：45.

[12] 赵越棋. 机器人实时控制平台研究与实现[D]. 中国科学院大学(中国科学院沈阳计算技术研究所)，2018.

[13] 倪敏，周怡廷页，杨继堂. μC/OS-Ⅱ的任务切换机理及中断调度优化[J]. 单片机与嵌入式系统应用，2003，3(10)：26-30.

[14] 徐阳，王锐，傅燕翔. 神经肌肉刺激器检测装置的研究[J]. 电子测试，2017(17).

[15] 李卫华. 汽车转弯过程中的防抱死制动系统的研究[D]. 燕山大学，2006.

[16] 廖银. 动态二进制翻译建模及其并行化研究[D]. 中国科学技术大学，2013.

[17] 李斌. 车载信息服务系统两项软件标准草案已经完成[J]. 信息技术与标准化，2012(8).

[18] 孙钟秀，操作系统教程[M]. 北京：高等教育出版社. 2003.

[19] 王远鹏. 提高实时操作系统的实时性能和可靠性策略[J]. 计算机光盘软件与应用，2012(16)：134-134.

[20] 祝旭晖. AUTOSAR系统建模方法的研究与实现[D]. 哈尔滨工业大学，2009.

[21] Zhw888888的博客. μC/OS-Ⅱ的任务切换机理及中断调度优化. [EB/OL]. (2010-06-21)[2019-3-14]. https://blog.csdn.net/zhw888888/article/details/5683201.

[22] 徐战亚. 可移植嵌入式导航平台关键技术研究[D]. 中国地质大学，2010.

[23] 张鸥. 智能网联汽车安全网关技术的研究与实现[D]. 电子科技大学，2018.

[24] 杜鸣皓. “自动驾驶”的现实挑战：尚待过法律和安全关[J]. 中国品牌，2017(05)：80-85.

[25] 走向希望的博客. 实时系统和分时系统. [EB/OL]. (2018-05-22)[2019-3-11] http://blog.sina.com.cn/s/blog_4b9eab320100y8c8.html.

[26] 电子电路. 基于MSP430F149的无线温湿度传输系统. [EB/OL]. (2013-07-14)[2019-3-17]. https://wenku.baidu.com/view/cab58900a8114431b90dd8d1.html.

[27] 计算机. 一种新型车路无线报站系统的设计与实现. [EB/OL]. (2012)[2019-3-17]. http://www.docin.com/p-64792695.html.

[28] 曹磊. 基于 LPC2478 炮兵信息处理设计与实现[D]. 电子科技大学,2013.
[29] 任慰. 以实时操作系统为中心的嵌入式系统平台化设计研究[D]. 华中科技大学,2013.
[30] 丁松涛. 基于 SD 卡存储的温湿度记录器的设计与实现[D]. 北京工业大学,2012.
[31] 张逢喆. 公共云计算环境下用户数据的隐私性与安全性保护[D]. 复旦大学,2010.
[32] 互联网文档资源. 公交站台自动报站系统的研究. [EB/OL]. (2017)[2019-3-17]. http://wenku.baidu.c.