AUTOSAR规范

与车用控制器软件开发

宋珂　王民　单忠伟　谭杨　编著

化学工业出版社

书名页

AUTOSAR规范
与车用控制器软件开发

宋珂　王民　单忠伟　谭杨　编著

化学工业出版社
·北京·

内容提要

本书共分10章，首先介绍了汽车电子控制系统以及AUTOSAR规范的基本概念；之后以AUTOSAR方法论为线索，通过示例开发过程介绍，全面展现了基于AUTOSAR工具链完成符合AUTOSAR规范的车用控制器软件开发的具体流程与方法；最后，剖析了AUTOSAR对道路车辆功能安全ISO 26262标准的支持情况，并对AUTOSAR与信息安全以及Adaptive AUTOSAR平台进行了介绍。本书以通俗易懂的语言和形象的图解展现了AUTOSAR中一些复杂的概念问题，精心设计的示例亦旨在以开发者的视角深度剖析AUTOSAR方法论的具体实施过程。

本书可作为汽车电子相关专业高年级本科生和研究生的参考书，也可作为汽车电子行业软件工程师学习参考的资料。

版权页

书名：AUTOSAR规范与车用控制器软件开发

作者：宋珂等编著

CIP号：第206980号

ISBN：978-7-122-32983-7

责任编辑：辛　田

出版发行：化学工业出版社（北京市东城区青年湖南街13号 100011）

购书咨询：010-64518888

售后服务：010-64518899

网址：http://www.cip.com.cn

版权所有　违者必究

序1

很高兴为本书作序。

汽车电子已成为汽车产品功能拓展与性能提升的重要技术支撑，而软件则是汽车电子的灵魂。对于汽车电子软件行业而言，AUTOSAR规范的应用打破了原有的汽车嵌入式系统软件开发模式，其快速提升软件质量及方便移植的特性降低了参与底层平台开发的门槛，对众多OEM厂商和Tier1而言可谓意义重大。自AUTOSAR规范诞生以来，至今亦不过十几载，而大规模应用更是近几年的事情。由于引入众多新的概念、新的工具、新的方法，AUTOSAR与传统的嵌入式软件开发模式差异较大，对于初学者来说都会感到有一点不适应。同时，目前与AUTOSAR相关的参考资料太少了，大多也都仅仅停留于AUTOSAR规范的解读层面，对于基于AUTOSAR工具链的开发过程详细讲解的书籍和资料可谓是极少，而有实战经验的开发工程师直接参与的著作就更少了。

上海捷能汽车技术有限公司自2009年成立以来，一直担负着为上汽集团的新能源汽车研发核心“三电”技术与产品的责任。由于立足于自主研发，软件体系能力的建立尤为迫切。为此，我们自公司成立之初就设置了负责软件开发的控制集成部门，并且在2011年前后开始学习和探索AUTOSAR 规范的落地应用。目前，上汽捷能已经完成了符合AUTOSAR规范的新能源汽车整车控制器VCU、电动机控制器MCU、电驱变速箱（EDU）控制器HCU等多个控制器的开发与量产，并建立了一支近200人专门从事控制软件开发的技术团队。国内的其他主流整车企业，以及一些零部件供应商也都已经开始为下一代符合AUTOSAR规范的“三电”系统ECU进行布局。

同济大学宋珂、上汽捷能资深软件工程师王民、同济大学单忠伟与ETAS谭杨合作撰写了《AUTOSAR规范与车用控制器软件开发》一书，分享了多位作者在AUTOSAR学习和开发中的经验，这对大家认识AUTOSAR规范，学习遵循AUTOSAR方法论的汽车软件开发过程提供了有益的参考。全书先从AUTOSAR规范相关概念着手进行讲解，然后基于实际事例介绍了符合AUTOSAR规范的车用控制器的软件开发方法。最后，分享了AUTOSAR与功能安全、信息安全等交叉领域的部分经验，还对下一代Adaptive AUTOSAR技术进行了介绍……这些都有助于汽车软件工程师更好地理解这些新概念和新系统，并对AUTOSAR的下一步发展方向有一个较为清晰的认识。

随着新能源和智能网联技术在汽车领域的应用发展，传统的分布式电子电气架构向新一代集中式电子电气架构发展的趋势愈发明显，与之相适应的软件将占有越来越重要的地位。快速走向市场的时间压力和越来越庞杂的功能开发需求将迫使更多的产品采用AUTOSAR架构，乃至下一代的Adaptive AUTOSAR架构。本书的出版对于想学习AUTOSAR的汽车电子专业学生和企业工程师而言都将是一件非常有意义的事情！

上汽乘用车技术中心副主任

上汽集团捷能公司总经理

教授级高级工程师

朱　军　博士

序2

如今，汽车电子技术在动力总成控制、底盘控制、车身控制以及车载信息娱乐系统等各个部分所占的比重越来越大，在整车成本中的占比也越来越高。随着汽车“电动化、网联化、智能化、共享化”的全面推进，几乎任何一项新技术的诞生都离不开汽车电子的身影。未来，汽车电子技术将成为汽车产品差异性的驱动力。ECU作为汽车电子控制系统的核心，其软件也变得日益复杂，传统的软件架构及开发模式已经不能适应日益复杂的汽车软件需求，此时AUTOSAR就是一个非常理想的解决方案。与传统ECU软件架构相比，AUTOSAR分层架构的高度抽象使得汽车嵌入式系统软、硬件耦合度大大降低。

AUTOSAR的应用源于德国，之后是美国、日本，现在已经到了中国。符合AUTOSAR规范的车用控制器开发在不久的未来将会成为行业趋势。

AUTOSAR方法论所提出的开发思路是一种基于工具链的开发，这与传统手写代码的方式具有很大的差异。工程师们在开始学习这一技术的时候需要花费大量的精力去熟悉整套开发流程。但是由于这一技术在中国刚刚起步，市面上几乎找不到合适的、针对性的参考书籍，因此一本好的AUTOSAR参考书籍是非常应景，非常必要的。

在仔细阅读了由同济大学宋珂、单忠伟以及上汽捷能王民与ETAS谭杨所著的《AUTOSAR规范与车用控制器软件开发》一书后，感觉耳目一新，这正是汽车电子行业内目前所急需的书籍。作者通过用心设计的例程详细讲解AUTOSAR规范的相关概念，揭示AUTOSAR方法论的具体实施方法，非常有助于读者去理解AUTOSAR的精髓。最后，本书还对AUTOSAR技

术进行了展望。这对于AUTOSAR规范在中国的推广应用具有重要意义！

国家“千人计划”专家

同济大学校学术委员会委员

汽车学院学术委员会主任

燃料电池汽车技术研究所所长

同济大学教授

章　桐　博士

序3

现代汽车是由电子元件和软件驱动的，它们在过去20年里驱动着创新，也将是未来10年汽车界创新的源动力。汽车已经成为人们设计的最复杂的软件系统之一，比如现代豪华车上运行着超过1亿行代码，基于复杂的车内网络在上百个ECU间进行数据交互。除了高复杂度之外，软件更承载着极其严酷的安全需求，软件的失效非常容易导致危险情况的发生。在这些安全需求的背后，信息安全近来也成为了一个焦点——没有信息安全就没有功能安全，因为不能保证软件不会以一些不安全的方式被非预期地篡改。

AUTOSAR组织经过15年的努力设计了一个完善的软件平台，它可以为应对软件复杂性问题提供重要的技术支撑，并且可以让Tier1和OEM更专注于软件功能的设计与开发。基于该平台，可以开发动力总成、底盘、车身和驾驶辅助系统等领域的ECU软件。它涵盖了构建现代ECU所需的实时调度、通信、诊断、存储管理、功能安全和信息安全等功能。如今，符合AUTOSAR规范的软件已用于数亿辆汽车和数十亿个ECU，仅ETAS就已经将其平台软件用于超过15亿个ECU，并以每周超过200万个新ECU的生产速度在增长。

由同济大学的宋珂和单忠伟、上海捷能汽车技术有限公司的王民以及ETAS的谭杨合作撰写的《AUTOSAR规范与车用控制器软件开发》一书，通过理论剖析和实践指导相结合的方式，深入介绍了AUTOSAR规范及其方法论的具体实施过程。对于参与汽车ECU设计或者开发的人员而言，非常值得一读。书中，作者以车灯控制器的开发示例诠释了基于AUTOSAR工具链、遵循AUTOSAR方法论，在AUTOSAR规范所定义的平台软件上进行功能开发的过程。

当下，汽车行业正在发生一场巨变，“电动化、网联化、智能化、共享化”引领着汽车创新的未来。为迎合这场变革带来的新需求，AUTOSAR组织定义了一个新的自适应平台。该平台提供了更灵活的、面向服务的架构来满足“新四化”过程中所面临的复杂计算和精准决策等难题。本书也介绍了AUTOSAR规范中的一些新概念以及自适应AUTOSAR平台相关的内容。

本书的出版可谓是非常及时，内容也十分充实，不仅立足于基础，还展望了AUTOSAR规范的未来发展方向。这对于有志于从事以软件驱动的汽车电子行业工作的学生或者工程师而言都将是一份宝贵的学习资料。

Dr. Nigel Tracey

Director-RTA Solutions

ETAS

前言

汽车电子技术已成为汽车各方面功能拓展、性能提升的重要技术支撑。随着汽车新能源化与智能化的逐步推进，汽车电子技术的功能需求将不断增加，控制软件也将变得越来越复杂。为了提升软件复用度，提高软件开发质量与效率并降低开发风险与成本，由全球汽车制造商、零部件供应商及其他半导体和软件系统公司联合建立了汽车开放系统架构联盟（AUTomotive Open System ARchitecture，AUTOSAR），并联合推出了一个开放的、标准化的汽车嵌入式系统软件架构——AUTOSAR规范。

AUTOSAR规范在国外的应用已经较为普遍和成熟，随着AUTOSAR规范的认可度越来越高，它有望成为整个汽车电子行业普遍使用的软件标准。近年来，随着国内一些企业纷纷加入新能源汽车“三电”相关控制器的研发，控制器正向开发需求不断增加，AUTOSAR规范在国内的应用也进入了一个高潮，基于AUTOSAR平台可以使得开发者更高效、更高质量地完成汽车嵌入式系统软件的开发。

本书中笔者以通俗易懂的语言、形象的图解展现了AUTOSAR中一些复杂的概念问题，并精心设计了一个示例作为本书的开发对象。笔者主要以ETAS AUTOSAR系统解决方案为基础，以AUTOSAR方法论为线索，详细介绍了基于AUTOSAR工具链完成车用控制器软件开发的具体流程与方法，并将基本概念融入开发过程介绍，加深读者的印象，提升读者的感性认识和认知水平。最后，还剖析了AUTOSAR对功能安全的支持情况，并对AUTOSAR与信息安全以及Adaptive AUTOSAR平台进行了介绍。

本书共分为10章。第1章介绍了汽车电子控制系统的发展史、应用

现状和基本构成，并提出了当下车用控制器软件所面临的问题。第2章介绍了AUTOSAR的基础理论知识，详细介绍了AUTOSAR分层架构、软件组件、虚拟功能总线、方法论及应用接口。第3章介绍了本书示例的开发需求、设计方案以及本书所采用的AUTOSAR系统解决方案，起到承上启下的作用；第4～8章详细介绍了AUTOSAR方法论的具体实施过程，以方法论为“纲”，各阶段配置开发为“目”，纲举目张，便于读者理解开发过程中每个阶段的作用，并学会AUTOSAR工具链的基本使用方法。其中，第4章主要讲述了使用Matlab/Simulink进行应用层软件组件开发以及符合AUTOSAR规范的代码和描述文件配置生成方法。第5章主要讲解了使用ETAS ISOLAR-A工具进行AUTOSAR系统级设计与配置的方法。第6章详细阐述了本书示例所涉及的基础软件模块和运行时环境的基本概念，以及基于ETAS RTA系列工具进行AUTOSAR ECU级开发的具体方法，包括CAN通信协议栈、ECU状态管理器、BSW模式管理器、运行时环境RTE、操作系统OS等常用模块。第7章则详细介绍了本书示例所用到的微控制器抽象层MCAL各模块的基本概念、配置及接口代码实现方法，基本覆盖了所有常用的MCAL模块。第8章介绍了AUTOSAR工程代码集成与调试方法，并展示了本书示例的开发结果。第9章和第10章主要介绍了AUTOSAR与功能安全、AUTOSAR与信息安全以及Adaptive AUTOSAR平台的相关内容，作为本书内容的拓展外延。

本书第1章～第7章由同济大学宋珂、单忠伟编写，第8章由ETAS谭杨编写，第9章与第10章由上海捷能汽车技术有限公司王民编写，书中示例由同济大学宋珂、单忠伟设计开发。全书由宋珂统稿，王民及ETAS ERS部门高级经理汤易负责审阅。

在本书编写过程中得到了ETAS公司、恩智浦半导体公司和MathWorks公司的支持！

本书适合具有一定嵌入式软件开发基础知识的读者阅读，可作为高等院校本科生、研究生学习AUTOSAR规范以及符合AUTOSAR规范的车用控

制器软件开发方法的参考书，也可以作为汽车电子行业软件工程师学习参考的资料。

本书中所有内容都经过ETAS公司、恩智浦半导体公司和MathWorks公司相关专家的审阅，且本书示例经过笔者亲自测试验证。但由于我们水平有限，书中难免会出现疏漏或不当之处，诚望读者批评和指正。

编著者

目录

第1章　汽车电子控制系统介绍
1.1　电子技术在汽车上的应用
1.1.1　汽车电子技术的发展历史
1.1.2　汽车电子技术的应用现状
1.2　汽车电子控制系统的基本构成
1.3　车用控制器软件标准（从OSEK到AUTOSAR）
1.4　本章小结
第2章　AUTOSAR规范基础理论
2.1　AUTOSAR的由来与发展历程
2.1.1　AUTOSAR的由来
2.1.2　AUTOSAR的原则及核心思想
2.1.3　AUTOSAR的发展历程及应用现状
2.2　AUTOSAR分层架构
2.2.1　AUTOSAR应用软件层
2.2.2　AUTOSAR运行时环境
2.2.3　AUTOSAR基础软件层
2.3　AUTOSAR软件组件
2.3.1　软件组件的数据类型
2.3.2　软件组件的端口与端口接口
2.3.3　软件组件的内部行为
2.4　AUTOSAR虚拟功能总线
2.5　AUTOSAR方法论
2.6　AUTOSAR应用接口
2.7　本章小结
第3章　本书示例及AUTOSAR系统解决方案介绍

3.1　本书示例介绍
3.1.1　示例开发需求介绍
3.1.2　示例总体方案设计
3.1.3　示例系统设计
3.1.4　示例系统AUTOSAR架构
3.2　ETAS AUTOSAR系统解决方案介绍
3.3　本书AUTOSAR系统解决方案介绍
3.4　本章小结
第4章　AUTOSAR软件组件级设计与开发
4.1　Matlab/Simulink与Embedded Coder工具简介
4.1.1　Matlab/Simulink工具简介
4.1.2　Embedded Coder工具简介
4.2　基于Matlab/Simulink的软件组件开发
4.2.1　Matlab/Simulink与AUTOSAR基本概念的对应关系
4.2.2　软件组件内部行为建模方法
4.2.3　AUTOSAR客户端/服务器机制的实现方法
4.3　软件组件代码及描述文件配置生成
4.3.1　求解器及代码生成相关属性配置
4.3.2　模型配置
4.3.3　AUTOSAR Properties配置
4.3.4　Simulink-AUTOSAR Mapping配置
4.3.5　符合AUTOSAR规范的代码及描述文件生成
4.4　在Simulink中导入软件组件描述文件——“自上而下”的工作流程
4.5　本章小结
第5章　AUTOSAR系统级设计与配置
5.1　ETAS ISOLAR-A工具简介
5.2　ETAS ISOLAR-A工具入门
5.2.1　ISOLAR-A安装方法

5.2.2　ISOLAR-A界面说明
5.3　基于ISOLAR-A的软件组件设计方法
5.3.1　AUTOSAR工程创建
5.3.2　数据类型定义
5.3.3　端口接口设计
5.3.4　软件组件设计
5.3.5　I/O硬件抽象层软件组件设计
5.3.6　软件组件模板生成
5.4　基于ISOLAR-A的系统级设计与配置方法
5.4.1　系统配置输入文件创建与导入
5.4.2　Composition SWC建立
5.4.3　系统配置
5.4.4　ECU信息抽取
5.5　本章小结
第6章　AUTOSAR ECU级开发之RTE与BSW（除MCAL外）
6.1　ETAS　RTA系列工具简介
6.1.1　RTA-BSW简介
6.1.2　RTA-RTE简介
6.1.3　RTA-OS简介
6.2　ETAS　RTA系列工具入门
6.2.1　RTA系列工具安装方法
6.2.2　RTA系列工具界面说明
6.3　CAN通信协议栈概念与配置方法介绍
6.3.1　CAN通信协议栈概念
6.3.2　CAN通信协议栈配置方法
6.4　EcuM模块概念与配置方法介绍
6.5　BswM模块概念与配置方法介绍
6.6　BSW模块代码生成

6.7　服务软件组件与应用层软件组件端口连接
6.8　RTE配置与代码生成
6.8.1　RTE Contract阶段生成
6.8.2　RTE配置
6.8.3　RTE Generation阶段生成
6.9　AUTOSAR操作系统概念与配置方法介绍
6.9.1　AUTOSAR操作系统概念
6.9.2　RTA-OS工程创建
6.9.3　AUTOSAR操作系统配置方法
6.9.4　RTA-OS工程编译
6.10　本章小结
第7章　AUTOSAR ECU级开发之MCAL
7.1　MCAL配置工具入门
7.1.1　MCAL配置工具安装方法
7.1.2　MCAL配置工具界面说明
7.1.3　MCAL配置工程创建方法
7.2　MCAL模块配置方法及常用接口函数介绍
7.2.1　Mcu模块
7.2.2　Gpt模块
7.2.3　Port模块
7.2.4　Dio模块
7.2.5　Adc模块
7.2.6　Pwm模块
7.2.7　Icu模块
7.2.8　Can模块
7.2.9　Base与Resource模块
7.3　MCAL配置验证与代码生成
7.4　本章小结

第8章　AUTOSAR工程代码集成与调试
8.1　AUTOSAR工程代码架构与集成方法介绍
8.2　代码编译链接
8.3　代码调试
8.3.1　单片机可执行文件下载
8.3.2　A型车灯调试现象
8.3.3　B型车灯调试现象
8.4　本章小结
第9章　AUTOSAR与功能安全
9.1　AUTOSAR对ISO 26262中支持部分的要求概述
9.1.1　ISO 26262对架构设计的要求
9.1.2　ISO 26262对硬件验证的要求
9.1.3　ISO 26262对通信验证的要求
9.1.4　ISO 26262对FFI的要求
9.1.5　ISO 26262对编码风格的要求
9.2　AUTOSAR中实现FFI的安全机制
9.2.1　AUTOSAR安全机制的存储空间分区
9.2.2　AUTOSAR安全机制的存储空间保护
9.2.3　AUTOSAR安全机制的程序流监控
9.2.4　AUTOSAR安全机制的E2E保护
9.3　本章小结
第10章　AUTOSAR技术展望
10.1　AUTOSAR与信息安全
10.1.1　密码协议栈
10.1.2　安全车载通信
10.2　Adaptive AUTOSAR平台
10.2.1　Adaptive AUTOSAR缘起
10.2.2　AP和CP

10.2.3　Adaptive AUTOSAR平台新概念介绍
10.3　本章小结
参考文献

第1章　汽车电子控制系统介绍

如今，电子技术在汽车中的应用日益广泛，汽车电子已成为汽车领域最热门的话题与研究方向之一。作为全书的引子，本章先介绍电子技术在汽车上的应用、汽车电子控制系统的基本构成以及车用控制器软件标准。

1.1　电子技术在汽车上的应用

1.1.1　汽车电子技术的发展历史

汽车电子技术的发展史是一段以电子技术发展为基础，以人们对汽车功能需求的日益增长为驱动力的发展史，其大致可以分为以下四个阶段。

第一个发展阶段：从20世纪50年代中期到70年代中期，这是汽车电子技术发展的起始阶段。在那时，一些汽车厂家开始研发一些单一的电子零部件，用来改善汽车上某些机械部件的性能，以及采用一些简单的电子设备来取代以前的机械部件，如整流器、电压调节器、交流发电机、晶体管无触点点火装置、电子喇叭、数字钟、汽车收音机等都是这一阶段出现的具有代表性的汽车电子装置。

第二个发展阶段：从20世纪70年代末期到80年代初期，以集成电路和16位以下的微处理器在汽车上的应用为标志，主要是开发汽车各系统专用的独立控制部分，电子装置被应用在某些机械装置无法解决的复杂控制功能方面。这期间最具代表性的是电子控制汽油喷射技术的发展和防抱死制动技术的成熟。该阶段涌现的其他汽车电子技术还包括自动门锁、高速警告系统、自动除霜控制、撞车预警传感器、电子正时、电子变速器、闭环排气控制、自动巡航控制、防盗系统等。

第三个发展阶段：从20世纪80年代中期到90年代初期，随着大规模集成电路技术的快速发展和微处理器在控制技术方面的应用，汽车电子技术迅速发展。此阶段主要是开发可以完成各种功能的综合系统，如集发动机控制与自动变速器控制为一体的动力传动控制系统、制动防抱死系统与驱动防滑转控制系统等。

第四个发展阶段：从20世纪90年代中期至今，随着计算机运算速度

和存取位数的提高以及车载网络与通信技术的迅速发展，车辆的智能控制和网络控制技术应运而生，它们给汽车赋予了更多的“想象力”。在当今汽车“电动化、网联化、智能化、共享化”的过程中，几乎任何一项新技术的诞生都离不开汽车电子技术的身影。

1.1.2 汽车电子技术的应用现状

目前，汽车电子技术主要应用于动力传动总成电子系统、底盘电子系统、车身电子系统、汽车通信与娱乐电子系统等。

（1）动力传动总成电子系统

①对于传统汽车而言，动力传动总成电子控制系统主要包括：发动机管理系统（Engine Management System，EMS）、自动变速器控制系统（Automatic Transmission Control System）等。

②对于新能源汽车而言，则主要包括：整车控制器（Vehicle Control Unit，VCU）、混合动力控制单元（Hybrid Control Unit，HCU）、驱动电动机控制器（Motor Control Unit，MCU）、电池管理系统（Battery Management System，BMS）、燃料电池发动机控制系统（Fuel Cell Engine Control System，FCECS）等。

它们主要是保证汽车动力系统在不同的工况下均能在最佳状态下运行，从而降低能耗，并简化驾驶员的操作，减轻驾驶员的劳动强度，以此来提高汽车的动力性、经济性和舒适性。

（2）底盘电子系统

汽车底盘由传动系统、行驶系统、转向系统和制动系统四大系统组成，随着电子技术在汽车中的广泛应用，使汽车底盘的控制正在快速地

向电子化、智能化和网络化方向发展，从而出现了许多汽车底盘电子控制系统。常用的控制系统如下。

①牵引力控制系统（Traction Control System，TCS）。

②电子稳定控制系统（Electronic Stability Control，ESC）。

③电控悬架系统（Electronic Control Suspension System，ECS）。

④定速巡航系统（Cruise Control System，CCS）。

⑤自适应巡航控制系统（Adaptive Cruise Control，ACC）。

⑥电动助力转向系统（Electric Power Steering，EPS）。

⑦防抱死制动控制系统（Anti-lock Braking System，ABS）。

⑧电子制动力分配系统（Electronic Brakeforce Distribution，EBD）。

⑨电子控制制动辅助系统（Electronic Brake Assist，EBA）。

⑩自动紧急制动系统（Autonomous Emergency Braking，AEB）。

⑪车道偏离预警系统（Lane Departure Warning，LDW）。

⑫车道保持辅助系统（Lane Keeping Assist，LKA）。

（3）车身电子系统

车身电子控制系统主要用于增强汽车的安全性与舒适性，常用的如下。

①自动采暖通风和空调系统（Heating，Ventilation and Air-Conditioning System，HVAC）。

②胎压检测系统（Tire Pressure Monitoring System，TPMS）。

③安全气囊系统（Supplemental Restraint System，SRS）。

④座椅位置调节系统（Seat Adjustment Position Memory System，SAMS）。

⑤雷达车距报警系统（Radar Proximity Warning System，RPWS）。

⑥倒车报警系统（Reverse Vehicle Alarm System，RVAS）。

⑦前部碰撞预警系统（Forward Collision Warning System，FCWS）。

⑧盲点监测系统（Blind Spot Detection，BSD）。

⑨停车辅助系统（Parking Assist System，PAS）。

⑩中央门锁控制系统（Central Locking Control System，CLCS）。

⑪防盗报警系统（Guard Against Theft and Alarm System，GATA）。

⑫自适应前照灯系统（Adaptive Front-lighting System，AFS）。

⑬夜视辅助系统（Night View Assist，NVA）。

⑭疲劳驾驶预警系统（Driver Fatigue Monitor System，DFMS）。

（4）汽车通信与娱乐电子系统

汽车通信与娱乐电子系统主要用于提升驾乘人员的舒适性，常用的如下。

①北斗导航系统（BeiDou Navigation Satellite System，BDS）。

②全球定位系统（Global Positioning System，GPS）。

③汽车音响（Amplifier）。

④收音机（Radio）。

⑤车载电话（Car Telephone，CT）。

⑥抬头显示（Head Up Display，HUD）。

⑦人机界面（Man Machine Interface，MMI）。

可见，电子技术在汽车中的应用越来越广泛，汽车电子控制技术已成为支撑现代汽车发展的关键技术之一。将来，随着汽车“电动化、网联化、智能化、共享化”进程的全面推进，电子技术在汽车中的应用将越来越广泛，汽车电子行业的前景尤为广阔。

1.2　汽车电子控制系统的基本构成

汽车电子控制系统主要由传感器（Sensor）、电子控制单元（Electronic Control Unit，ECU）和执行器（Actuator）组成（图1.1），对被控对象（Controlled Object）进行控制。

图1.1　汽车电子控制系统的基本构成

（1）传感器

为信号输入装置，其作为汽车电子控制系统的信息源。传感器用来检测和采集各种信息，如温度、压力、转速等，并通过一定转换装置将一些非电量信号（物理量、化学量等）转换为电信号传给电子控制单元。

（2）电子控制单元（ECU）

也可称为汽车嵌入式系统（Automotive Embedded System，AES），是汽车电子控制系统的核心。ECU对传感器的信号进行处理，通过控制算法向执行器发出控制指令。电子控制单元一般由硬件和软件两部分组成，硬件部分主要由微控制器（Microcontroller，MCU）及外围电路组成；软件部分主要包括硬件抽象层（Hardware Abstraction Layer，HAL）、嵌入式操作系统及底层软件和应用软件层。

（3）执行器

执行器为执行某种控制功能的装置，用于接收来自ECU的控制指

令，并对控制对象实施相应的操作。

1.3　车用控制器软件标准（从OSEK到AUTOSAR）

为了迎合汽车高精度、高实时性、高可靠性控制的需要，嵌入式实时操作系统（Real Time Operating System，RTOS）逐渐在ECU中使用。与此同时，由于不同实时操作系统间应用程序接口（Application Programming Interface，API）的各不相同，出现应用程序的移植性差等问题。于是，在1993年德国汽车工业界提出了OSEK（Offene Systeme und deren Schnittstellen für die Elektronik im Kraftf其英语全称为Open Systems and the Corresponding Interfaces for Automotive Electronics，中文名称为汽车电子开放式系统及其接口标准。该体系最初得到了宝马、博世、戴姆勒-克莱斯勒、西门子、大众等公司的支持。1994年，随着标致和雷诺加入该体系，他们将法国汽车工业使用的汽车分布式执行标准（Vehicle Distributed eXecutive，VDX）也纳入该体系，并在1995年的研讨会上得到共识，从而产生了OSEK/VDX标准。

OSEK/VDX标准包括以下七个部分。

①OSEK/VDX操作系统规范（OSEK Operating System，OSEK OS）。

②OSEK/VDX通信规范（OSEK Communication，OSEK COM）。

③OSEK/VDX网络管理规范（OSEK Network Management，OSEK NM）。

④OSEK/VDX实现语言规范（OSEK Implementation Language，OSEK IL）。

⑤OSEK/ORTI运行时接口规范（OSEK Run Time Interface）。

⑥OSEK-Time时间触发操作系统规范（OSEK Time-Triggered Operating System）。

⑦OSEK FTCom容错通信规范（OSEK Fault-Tolerant Communication）。

OSEK/VDX标准在一定程度上使得应用层软件与底层软件分离，提升了应用软件的可移植性；其次，使用符合OSEK/VDX标准的嵌入式操作系统可以提高代码的复用率，提升开发效率、降低开发成本。但由于不同整车企业和零部件供应商缺乏兼容性工具，使得开发者需要花费大量的时间在基础软件的实现和优化上，并且对于新的需求，需要花费大量的精力调整软件的接口。总之，OSEK/VDX标准还是没能解决跨平台化高效地进行嵌入式系统软件的移植。

在之后的一段时间里，随着汽车分布式嵌入式系统软件复杂度的迅速增长，汽车工业界开始逐步探索从原有的以硬件设计和组件驱动为主的设计方式向以需求设计和功能驱动为主的系统开发方法转变。在该进程中，比较有代表性的是EAST-EEA项目，它是ITEA（International Test and Evaluation Association）资助的面向汽车领域嵌入式系统架构的研究项目，其目标是通过建立面向汽车工业的通用嵌入式系统架构，实现接口的标准化并提升复杂系统开发的质量与效率。最终，EAST-EEA项目定义了一个分层的软件架构，并提出了一个“中间件”的层次来提供支持嵌入式系统模块在不同平台之间移植的接口和服务；并且，EAST-EEA项目还定义了公共的架构描述语言（Architecture Description Language，ADL）。EAST-EEA项目的研究成果可谓是AUTOSAR规范的雏形。

1.4　本章小结

本章首先回顾了汽车电子技术的发展史，并详细介绍了电子技术在汽车中的应用现状，进而分析了汽车电子控制系统的基本构成。之后，介绍了车用控制器软件标准——OSEK标准，并揭示了车用控制器软件开发所面临的问题与挑战。通过本章的学习，可以较为全面地认识汽车电子控制系统的发展与应用现状、基本构成以及软件开发标准。

第2章　AUTOSAR规范基础理论

AUTOSAR规范作为汽车嵌入式系统软件的通用性规范，在软件架构、软件开发流程等方面都定义了众多新概念，掌握这些理论知识是进行符合AUTOSAR规范的软件开发的基础。所以，本章从AUTOSAR的由来及发展历程着手，详细介绍AUTOSAR规范中三块主要内容，即分层架构、方法论与应用接口，并对其中软件组件与虚拟功能总线的概念进行详细剖析。

2.1 AUTOSAR的由来与发展历程

2.1.1 AUTOSAR的由来

如前所述，电子技术在动力总成控制、底盘控制、车身控制以及车载信息娱乐系统等各个部分所占的比重越来越大，所占的整车成本也越来越高。电子技术已悄然成为汽车各方面功能拓展和性能提升的重要技术支撑。

由于汽车电子硬件系统的多样性，ECU软件的开发受到硬件系统的制约，每当需要更新硬件时，都会导致ECU软件重新编写或大规模修改，之后还要进行一系列测试，从而导致了高昂的研发费用与漫长的研发周期。

目前，汽车电子网络正向多总线混合网络互联方向发展；电控系统硬件正向专业化、高集成度、高性能方向发展，其软件架构也正向模块化、平台化、标准化方向发展。并且，未来随着汽车新能源化和智能化的普及，以及对于一些非功能需求的增加，汽车电子/电气系统的复杂度也将进一步提升。这都将进一步导致新产品开发周期、成本的急剧增加。整车厂为了降低汽车控制软件开发的风险，于是开始寻找提高软件复用度的方法。

为解决上述问题，基于先前EAST-EEA项目的研究成果，在2003年，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了汽车开放系统架构联盟（AUTomotive Open System ARchitecture，AUTOSAR），并联合推出了一个开放化的、标准化的汽车嵌入式系统软件架构——AUTOSAR规范。如图2.1所示，与传统ECU软件架构相比，AUTOSAR分层架构的高度抽象使得汽车嵌入式系统软硬件耦合度大大降低。

图2.1　传统软件架构与AUTOSAR架构对比

AUTOSAR规范的出现，将带来如下主要优势：

①有利于提高软件复用度，尤其是跨平台的复用度；

②便于软件的交换与更新；

③软件功能可以进行先期架构级别的定义和验证，从而能减少开发错误；

④减少手工代码量，减轻测试验证负担，提高软件质量；

⑤使用一种标准化的数据交换格式，方便各公司之间的合作交流等。

这些优势对将来愈发复杂的汽车嵌入式系统软件的开发过程可谓是大有裨益，在保证软件质量的同时，可以大大降低开发的风险与成本。

2.1.2　AUTOSAR的原则及核心思想

AUTOSAR联盟自成立至今，一直提倡“在标准上合作，在实现上竞争”的原则，标准大家共同制定，但具体的实现方法是由各公司自己去

探索的。其核心思想在于“统一标准、分散实现、集中配置”。“统一标准”是为了给各厂商提供一个开放的、通用的平台；“分散实现”要求软件系统高度的层次化和模块化，同时还要降低应用软件与硬件平台之间的耦合；不同的模块可以由不同的公司去完成开发，但要想完成最终软件系统的集成，就必须将所有模块的配置信息以统一的格式集中整合并管理起来，从而配置生成一个完整的系统，这就是“集中配置”。

采用AUTOSAR将为OEM（主机厂）带来很大的好处，使得其对于软件采购和控制拥有更灵活和更大的权利。因为AUTOSAR不仅在软件的功能上、接口上进行了一系列的标准化，还提出了一套规范化的开发流程与方法，这就使得能有更多的软件供应商进入汽车电子行业，大家都遵循同一个标准去开发，最终比的是产品的功能和质量。

2.1.3 AUTOSAR的发展历程及应用现状

AUTOSAR联盟从2003年成立至今，成员队伍不断壮大，标准内容日臻完美。AUTOSAR联盟成员如图2.2所示，可见AUTOSAR联盟成员按等级分为核心成员（Core Partner）、高级成员（Premium Member）以及合作成员三类正式成员，基本涵盖了世界上各大著名整车厂、零部件公司、半导体公司以及软件工具提供商。

图2.2　AUTOSAR联盟成员

目前，AUTOSAR平台最新版为4.3.1。为了迎合未来汽车智能化、网联化的需求，AUTOSAR联盟推出了一个全新的平台——自适应AUTOSAR平台（AUTOSAR Adaptive Platform，AP），并将现有平台更名为经典AUTOSAR平台（AUTOSAR Classic Platform，CP），AUTOSAR官网（https：//www.AUTOSAR.org/）也进行了更新，给人一种耳目一新的感觉。

由于AUTOSAR Adaptive Platform配套工具链还没发布，并且本书以介绍AUTOSAR基本理论知识与AUTOSAR方法论具体实现方法为主，所以，采用AUTOSAR Classic Platform，并且主要使用4.2.2和4.0.3版本。

AUTOSAR规范在国外的应用相较于国内更早、更普遍、更成熟。大众、博世、通用、德尔福、菲亚特等公司已将符合AUTOSAR规范的软件应用于它们的ECU产品。

德国大众集团与MathWorks、Elektrobit（EB）等公司联合开发了符

合AUTOSAR规范的车身舒适控制系统，并应用于帕萨特车型。

玛涅蒂玛瑞利公司将AUTOSAR应用于菲亚特汽油发动机平台，并进行了硬件在环测试和不同工况下2万千米行程的实车测试。此外，他们还将AUTOSAR运用于车灯控制、动力总成控制等电控系统。

ETAS公司成功将宝马5系发动机管理系统开发为符合AUTOSAR规范的控制系统。开发人员利用ASCET进行软件组件开发，管理软件组件端口及其运行实体；使用RTA-OS开发AUTOSAR操作系统，配置、划分、管理任务；应用RTA-RTE连接应用层和基础软件层；代码集成后，进行了硬件在环仿真，结果表明与传统开发方法相比复杂度降低50%。整个博世集团已将AUTOSAR架构应用于自适应巡航系统ECU、动力总成系统ECU、底盘控制系统ECU和车身控制模块（Body Control Module，BCM）的开发，并且今后将运用于更多的ECU开发过程。

近年来，随着国内新能源汽车相关控制器正向开发需求的增长，AUTOSAR规范在国内越来越受到大家的关注，并且应用需求也越来越大。目前，上汽、北汽等国内主流整车厂以及一些零部件供应商都开始致力于符合AUTOSAR规范的车用控制器软件开发。AUTOSAR规范也有望成为未来整个汽车电子行业所普遍使用的软件标准。

2.2 AUTOSAR分层架构

AUTOSAR规范主要包括分层架构、方法论和应用接口三部分内容。其中，分层架构是实现软硬件分离的关键，它使汽车嵌入式系统控制软件开发者摆脱了以往ECU软件开发与验证时对硬件系统的依赖。

在AUTOSAR分层架构中，汽车嵌入式系统软件自上而下分别为应用软件层（Application Software Layer，ASW）、运行时环境（Runtime Environment，RTE）、基础软件层（Basic Software Layer，BSW）和微控制器（Microcontroller），如图2.3所示。为保证上层与下层的无关性，在通常情况下，每一层只能使用下一层所提供的接口，并向上一层提供相应的接口。

图2.3 AUTOSAR分层架构

2.2.1 AUTOSAR应用软件层

应用软件层（Application Software Layer，ASW）包含若干个软件组件（Software Component，SWC），软件组件间通过端口（Port）进行交互。每个软件组件可以包含一个或者多个运行实体

（Runnable Entity，RE），运行实体中封装了相关控制算法，其可由RTE事件（RTE Event）触发。

2.2.2 AUTOSAR运行时环境

运行时环境（Runtime Environment，RTE）作为应用软件层与基础软件层交互的桥梁，为软硬件分离提供了可能。RTE可以实现软件组件间、基础软件间以及软件组件与基础软件之间的通信。RTE封装了基础软件层的通信和服务，为应用层软件组件提供了标准化的基础软件和通信接口，使得应用层可以通过RTE接口函数调用基础软件的服务。此外，RTE抽象了ECU之间的通信，即RTE通过使用标准化的接口将其统一为软件组件之间的通信。由于RTE的实现与具体ECU相关，所以必须为每个ECU分别实现。

2.2.3 AUTOSAR基础软件层

基础软件层（Basic Software Layer，BSW）又可分为四层，即服务层（Services Layer）、ECU抽象层（ECU Abstraction Layer）、微控制器抽象层（Microcontroller Abstraction Layer，MCAL）和复杂驱动（Complex Drivers），如图2.4所示。

图2.4 AUTOSAR基础软件层

上述各层又由一系列基础软件组件构成，包括系统服务

（System Services）、存储器服务（Memory Services）、通信服务（Communication Services）等，如图2.5所示。它们主要用于提供基础软件服务，包括标准化的系统功能和功能接口。

图2.5 AUTOSAR基础软件层结构

（1）服务层

服务层（Services Layer）提供了汽车嵌入式系统软件常用的一些服务，其可分为系统服务（System Services）、存储器服务（Memory Services）以及通信服务（Communication Services）三大部分。提供包括网络通信管理、存储管理、ECU模式管理和实时操作系统（Real Time Operating System，RTOS）等服务。除了操作系统外，服务层的软件模块都是与ECU平台无关的。

（2）ECU抽象层

ECU抽象层（ECU Abstraction Layer）包括板载设备抽象（Onboard Devices Abstraction）、存储器硬件抽象（Memory Hardware Abstraction）、通信硬件抽象（Communication Hardware Abstraction）和I/O硬件抽象（Input/Output Hardware Abstraction）。该层将ECU结构进行了抽象，

负责提供统一的访问接口，实现对通信、存储器或者I/O的访问，从而不需要考虑这些资源是由微控制器片内提供的，还是由微控制器片外设备提供的。该层与ECU平台相关，但与微控制器无关，这种无关性正是由微控制器抽象层来实现的。

（3）微控制器抽象层

微控制器抽象层（Microcontroller Abstraction Layer，MCAL）是实现不同硬件接口统一化的特殊层。通过微控制器抽象层可将硬件封装起来，避免上层软件直接对微控制器的寄存器进行操作。微控制器抽象层包括微控制器驱动（Microcontroller Drivers）、存储器驱动（Memory Drivers）、通信驱动（Communication Drivers）以及I/O驱动（I/O Drivers），如图2.6所示。

图2.6　微控制器抽象层

（4）复杂驱动层

由于对复杂传感器和执行器进行操作的模块涉及严格的时序问题，难以抽象，所以在AUTOSAR规范中这部分没有被标准化，统称为复杂驱动（Complex Drivers）。

2.3 AUTOSAR软件组件

软件组件（SWC）不仅仅是应用层的核心，也是一些抽象层、复杂驱动层等实现的载体。由于软件组件包含的概念较多，这里单独介绍AUTOSAR软件组件相关概念，这是后期进行应用层、抽象层等开发的基础。

AUTOSAR软件组件大体上可分为原子软件组件（Atomic SWC）和部件（Composition SWC）。其中，部件可以包含若干原子软件组件或部件。原子软件组件则可根据不同用途分为以下几种类型：

①应用软件组件（Application SWC）；

②传感器/执行器软件组件（Sensor/Actuator SWC）；

③标定参数软件组件（Parameter SWC）；

④ECU抽象软件组件（ECU Abstraction SWC）；

⑤复杂设备驱动软件组件（Complex Device Driver SWC）；

⑥服务软件组件（Service SWC）。

应用软件组件（Application SWC）主要用于实现应用层控制算法。

传感器/执行器软件组件（Sensor/Actuator SWC）用于处理具体传感器/执行器的信号，可以直接与ECU抽象层交互。

标定参数软件组件（Parameter SWC）主要提供标定参数值。

ECU抽象软件组件（ECU Abstraction SWC）提供访问ECU具体I/O的能力。该软件组件一般提供引用C/S接口的供型端口，即Server端，由其他软件组件（如传感器/执行器软件组件）的需型端口（Client端）调用。此外，ECU抽象软件组件也可以直接和一些基础软件进行交互。

复杂设备驱动软件组件（Complex Device Driver SWC）推广了ECU抽象软件组件，它可以定义端口与其他软件组件通信，还可以与ECU硬件直接交互。所以，该类软件组件灵活性最强，但由于其和应用对象强相关，从而导致其可移植性较差。

服务软件组件（Service SWC）主要用于基础软件层，可通过标准接口或标准AUTOSAR接口与其他类型的软件组件进行交互。

需要指出的是，上述这些软件组件有的仅仅是概念上的区分，从具体实现及代码生成角度而言都是相通的。下面将详细介绍AUTOSAR软件组件的几个重要概念：数据类型、端口、端口接口以及内部行为。

2.3.1 软件组件的数据类型

AUTOSAR规范中定义了如下三种数据类型（Data Type）：

①应用数据类型（Application Data Type，ADT）；

②实现数据类型（Implementation Data Type，IDT）；

③基础数据类型（Base Type）。

应用数据类型（Application Data Type，ADT）是在软件组件设计阶段抽象出来的数据类型，用于表征实际物理世界的量，是提供给应用层使用的，仅仅是一种功能的定义，并不生成实际代码。

实现数据类型（Implementation Data Type，IDT）是代码级别的数据类型，是对应用数据类型的具体实现；它需要引用基础数据类型（Base Type），并且还可以配置一些计算方法（Compute Method）与限制条件（Data Constaint）。

在AUTOSAR中，对于Application Data Type没有强制要求使用，用户可以直接使用Implementation Data Type。若使用了Application Data Type，则必须进行数据类型映射

（Data Type Mapping），即将Application Data Type与Implementation Data Type进行映射，从而来对每个Application Data Type进行具体实现。

2.3.2 软件组件的端口与端口接口

软件组件的端口根据输入/输出方向可分为需型端口（Require Port，RPort）与供型端口（Provide Port，PPort），在AUTOSAR 4.1.1标准中又提出了供需端口（Provide and Require Port，PRPort）。

①需型端口：用于从其他软件组件获得所需数据或者所请求的操作。

②供型端口：用于对外提供某种数据或者某类操作。

③供需端口：兼有需型端口与供型端口的特性。

需型端口可以和供型端口连接。如图2.7所示，软件组件SWC1有一个需型端口（R）和一个供型端口（P），其中需型端口与SWC2的供型端口相连，它们之间的交互关系通过连线箭头表示，由SWC2的供型端口指向SWC1的需型端口。SWC3具有一个供需端口，它可被认为自我相连。

图2.7　AUTOSAR软件组件端口

由于端口仅仅定义了方向，所以AUTOSAR中用端口接口（Port Interface）来表征端口的属性，端口接口主要有如下几种类型：

①发送者-接收者接口（Sender-Receiver Interface，S/R）；

②客户端-服务器接口（Client-Server Interface，C/S）；

③模式转换接口（Mode Switch Interface）；

④非易失性数据接口（Non-volatile Data Interface）；

⑤参数接口（Parameter Interface）；

⑥触发接口（Trigger Interface）。

其中，最常用的端口接口是发送者-接收者接口（Sender-Receiver Interface，S/R）与客户端-服务器接口（Client-Server Interface，C/S）。如图2.8所示，软件组件SWC1具有两个端口，

其中一个引用的端口接口类型为发送者-接收者（S/R）接口，另一个引用的端口接口类型为客户端-服务器（C/S）接口。从中也可以看出，对于引用发送者-接收者接口的一组端口而言，需型端口为接收者（Receiver），供型端口为发送者（Sender）。对于引用客户端-服务器接口的一组端口而言，需型端口为客户端（Client），供型端口为服务器（Server）。

图2.8　AUTOSAR软件组件端口接口

下面详细讨论发送者-接收者接口（Sender-Receiver Interface，S/R）与客户端-服务器接口（Client-Server Interface，C/S）的特性。

（1）发送者-接收者接口

发送者-接收者接口用于数据的传递关系，发送者发送数据到一个或多个接收者。该类型接口中定义了一系列的数据元素

（Data Element，DE），这些数据元素之间是相互独立的。如图2.9所示，该发送者-接收者接口SR_Interface中定义了两个数据元素，名字分别为DE_1与DE_2，并且需要为每个数据元素赋予相应的数据类型。

图2.9 发送者-接收者接口定义

需要指出的是，一个软件组件的多个需型端口、供型端口、供需端口可以引用同一个发送者-接收者接口，并且它们可以使用该接口中所定义的任意一个或者多个数据元素，而并不一定使用所有数据元素。

（2）客户端-服务器接口

客户端-服务器接口用于操作（Operation，OP），即函数调用关系，服务器是操作的提供者，多个客户端可以调用同一个操作，但同一个客户端不能调用多个操作。客户端-服务器接口定义了一系列操作（Operation），即函数，它（们）由引用该接口的供型端口所在的软件组件来实现，并提供给引用该接口的需型端口所在的软件组件调用。如图2.10所示，该客户端-服务器接口CS_Interface中定义了两个操作OP_1与OP_2，对于每一个操作需要定义相关参数及其方向，即函数的形参。

图2.10　客户端-服务器接口定义

需要注意的是，每个端口只能引用一种接口类型，并且引用相同端口接口类型的端口才可以进行交互。

2.3.3　软件组件的内部行为

软件组件的内部行为（Internal　Behaviour，IB）如图2.11所示，其主要包括：

图2.11　软件组件的内部行为

①运行实体（Runnable Entity，RE）；

②运行实体的RTE事件（RTE Event）；

③运行实体与所属软件组件的端口访问（Port Access）；

④运行实体间变量（Inter Runnable Variable，IRV）。

（1）运行实体

运行实体（Runnable Entity，RE）是一段可执行的代码，其封装了一些算法。一个软件组件可以包含一个或者多个运行实体。

（2）运行实体的RTE事件

每个运行实体都会被赋予一个RTE事件（Trigger Event），即RTE事件（RTE Event），这个事件可以引发这个运行实体的执行。对于RTE事件可以细分为很多种类，这将在后续章节介绍软件组件的内部行为设计时结合工具进行介绍。较常用的RTE事件有以下几种：

①周期性（Periodic）事件，即Timing Event；

②数据接收事件（Data-received Event）；

③客户端调用服务器事件（Server-call Event）。

如图2.12所示，其中Runnable_1、Runnable_2和Runnable_3分别采用了Timing Event、Data-received Event以及Server-call Event。

图2.12　运行实体的RTE事件示意

（3）运行实体与所属软件组件的端口访问

运行实体与所属软件组件的端口访问（Port Access）是和端口所引用的端口接口类型密切相关的。

对于S/R通信模式，可分为显示（Explicit）和隐式（Implicit）两种模式。若运行实体采用显示模式的S/R通信方式，数据读写是即时的；当多个运行实体需要读取相同的数据时，若能在运行实体运行之前先把数据读到缓存中，在运行实体运行结束后再把数据写出去，则可以改善运行效率，这就是隐式模式。显示模式与隐式模式的对比如图2.13所示，可见后者的实现方式中，会在运行实体被调用之前读数据，在运行结束后写数据。

图2.13　显式模式与隐式模式的对比

对于C/S通信模式，可分为同步（Synchronous）和异步（Asynchronous）两种模式，它们的对比如图2.14所示。

图2.14　同步模式和异步模式的对比

（4）运行实体间变量

运行实体间变量（Inter Runnable Variable，IRV）即两个运行实体之间交互的变量，如图2.15所示。

图2.15　运行实体间变量示意

2.4 AUTOSAR虚拟功能总线

若从整车级别去看待整车上所有的功能模块，即软件组件的架构，它们之间的通信形式主要涉及以下两种：

①在单个ECU内部的通信（Intra-ECU Communication）；

②在多个ECU之间的通信（Inter-ECU Communication）。

如果使用传统的系统设计方法，则会带来一个问题，即在定义整车级别的应用层软件架构的时候会受到具体实现手段的束缚，这主要体现在与底层软件的接口。AUTOSAR为了实现一种“自顶向下”的整车级别的软件组件定义，提出了虚拟功能总线（Virtual Function Bus，VFB）的概念。VFB可以使得负责应用层软件的开发人员不用去关心一个软件组件最终在整车中的哪个ECU中具体实现，即使得应用软件的开发可以独立于具体的ECU开发。从而，可以让应用软件开发人员专注于应用软件组件的开发。

VFB是AUTOSAR提供的所有通信机制的抽象。通过VFB，无论软件组件使用的是在ECU内部的通信还是在ECU之间的通信，对于应用软件的开发者而言，没有本质区别。内部通信与外部通信的区别只有等到系统级设计与配置阶段，将软件组件分配到不同的ECU之后才会体现出来。最终，VFB的真实通信实现可以由RTE和基础软件来保证，所以，RTE是AUTOSAR VFB的具体实现。

通过对通信机制的抽象，可以使得当一个系统的软件组件之间的通信关系确定之后，通过VFB就可以在开发前期将它们虚拟集成完成系统仿真与测试工作。

2.5 AUTOSAR方法论

AUTOSAR方法论（AUTOSAR Methodology）中车用控制器软件的开发涉及系统级、ECU级和软件组件级。系统级主要考虑系统功能需求、硬件资源、系统约束，然后建立系统架构；ECU级根据抽象后的信息对ECU进行配置；系统级和ECU级设计的同时，伴随着软件组件级的开发。上述每个环节都有良好的通信接口，并使用统一的arxml（AUTOSAR Extensible Markup Language）描述文件，以此构建了AUTOSAR方法论。AUTOSAR方法论中“自顶向下”的软件组件设计与VFB实现方法示意如图2.16所示，而对于单个ECU内部的系统实现方法示意如图2.17所示。

图2.16　AUTOSAR方法论——“自顶向下”的软件组件设计与VFB实现方法示意

图2.17　AUTOSAR方法论——单个ECU内部的系统实现方法示意

在开发之前，需要先编写系统配置输入描述文件，其包含以下三部分内容。

①软件组件描述（SW-Component Description）：包含系统中所涉及的软件组件的接口信息，例如数据类型、端口接口、端口等。

②ECU资源描述（ECU Resource Description-HW only）：包含系统中每个ECU所需要的处理器及其外设、传感器、执行器等信息。

③系统约束描述（System Constraint Description）：包含总线信号、软件组件间的拓扑结构和一些映射关系等信息。

基于上述系统配置输入描述文件，系统配置根据ECU资源和时序要求，将软件组件映射到对应的ECU上，生成系统配置描述文件（System Configuration Description）。系统配置描述文件中包含了设计过程中非常重要的一个描述——系统通信矩阵，其描述了网络中所有运

行的数据帧及其对应的时序和内容。

从系统级到ECU级的过渡操作是指ECU信息抽取（ECU Extract）。在系统配置阶段已经将每个ECU所包含的所有软件组件、网络通信等信息封装好，ECU信息抽取阶段只需将待配置ECU信息抽取出来即可，服务于之后的ECU配置。

ECU配置过程主要是对RTE和BSW的配置。在RTE配置阶段，需要将软件组件的运行实体映射到相应的操作系统任务；在BSW配置阶段，需要详细配置BSW层中所需要用到的模块，一般有操作系统、通信服务、ECU抽象层和微控制器抽象层等。依据ECU配置信息生成BSW和RTE代码，再结合软件组件级实现的应用代码，最终进行代码集成，编译链接，生成单片机可执行文件。

2.6 AUTOSAR应用接口

AUTOSAR规范中，将不同模块间通信的接口主要分为以下三类：

①AUTOSAR接口（AUTOSAR Interface）；

②标准AUTOSAR接口（Standardized AUTOSAR Interface）；

③标准接口（Standardized Interface）。

AUTOSAR接口（AUTOSAR Interface）属于应用接口，是从软件组件的端口衍生来的通用接口，描述数据或者服务。它由RTE提供给软件组件，可以作为软件组件间通信的接口，也可以作为软件组件与I/O硬件抽象层或复杂设备驱动层间的接口。AUTOSAR接口非标准，可自定义，但在AUTOSAR规范中目前已对车身、底盘及动力传动系统控制领域的应用接口做了一些标准化工作。

标准AUTOSAR接口（Standardized AUTOSAR Interface）是一种特殊的AUTOSAR接口，在AUTOSAR规范中有明确的定义。由RTE向软件组件提供BSW中的服务，如存储器管理、ECU状态管理、“看门狗”管理等。

标准接口（Standardized Interface）在AUTOSAR规范中以C语言中API的形式明确定义。主要用于ECU上的BSW各模块间、RTE和操作系统间、RTE和通信模块间，应用软件组件不可访问。

上述三种接口的示意如图2.18所示。

图2.18　AUTOSAR软件构架的接口示意

2.7 本章小结

本章首先介绍了AUTOSAR规范的由来及其发展历史，并详细剖析了AUTOSAR规范中所蕴含的基本概念。其中，在分析AUTOSAR分层架构、AUTOSAR方法论以及AUTOSAR应用接口的同时，对AUTOSAR软件组件和虚拟功能总线的概念单独进行了解析。通过本章的学习，可以较好地理解AUTOSAR规范中的基本概念，这些都是之后进行符合AUTOSAR规范的车用控制器软件开发的基础。

第3章　本书示例及AUTOSAR系统解决方案介绍

在上一章介绍AUTOSAR相关理论知识的基础上，本书后面几章将以一个具体示例开发过程为例详细介绍符合AUTOSAR规范的车用控制器软件开发的基本方法。为便于读者对本书示例及AUTOSAR系统解决方案有一个整体性的认识，本章将先介绍本书示例的开发需求与设计方案，进而介绍本书中示例开发所使用的AUTOSAR系统解决方案。

3.1 本书示例介绍

3.1.1 示例开发需求介绍

“南极洲”某整车厂有A型和B型两种车型，其中A型为低端车型，B型为高端车型。现需要为它们设计两款车灯控制器。

①A车型：车灯开关打开，车灯就亮。

②B车型：车灯开关打开，车灯会根据外界光强情况自动调节亮度。

并且，在车灯开关关闭时间长于5分钟后，关闭车灯控制器的CAN通信；一旦检测到车灯开关打开，则立即恢复CAN通信。

说明：本书示例与实际相比做了一定假设和简化，仅为尽可能全面地展现AUTOSAR相关概念及方法论的具体实施方法而设计。

3.1.2 示例总体方案设计

对于A车型车灯控制器：用数字输入（Digital Input，DI）检测车灯开关信号，用数字输出（Digital Output，DO）直接控制车灯；用模数转换（Analog-to-Digital Converter，ADC）采集DO输出，作为车灯状态检测，并将检测状态以及车灯类型通过CAN报文发到CAN网络上。

对于B车型车灯控制器：用数字输入（Digital Input，DI）检测车灯开关信号，结合通过CAN报文接收到的外界光强信号，使用脉冲宽度调制（Pulse Width Modulation，PWM）输出不同占空比的脉冲来调节车灯亮度；用输入捕获单元（Input Capture Unit，ICU）采集PWM输出占空比，作为车灯状态检测，并将检测状态以及车灯类型通过CAN报文发到CAN网络上。

3.1.3 示例系统设计

根据系统总体方案设计，可以进行本书示例的系统设计，这里主要介绍CAN通信矩阵设计、系统软件架构设计以及目标ECU软件组件设计。

（1）CAN通信矩阵设计

由于本书示例采用基于CAN总线的系统，所以需要进行CAN通信矩阵设计，本系统涉及三个ECU：SensorECU、LightECU、DisplayECU，包含两帧CAN报文，具体设计如表3.1所示。

表3.1 CAN通信矩阵设计

ID	报文名称	方向	节点	信号	长度/位	信号值描述
0x01	Light_Intensity	Rx	SensorECU→LightECU	LightIntensity	8	外界光线强度，对应 PWM 波输出占空比 －1：弱→占空比 100％ 0：中→占空比 50％ 1：强→占空比 0
0x02	Light_State	Tx	LightECU→ DisplayECU	LightType	8	1：A 型 2：B 型
				LightState	8	A 型：0 表示关，1 表示开 B 型：直接显示 PWM 波占空比值

（2）系统软件架构设计

为充分利用AUTOSAR分层架构模块化复用的优势，将A型与B型车灯控制器采用同一个软件架构，其软件架构设计如图3.1所示，其中各软件组件的描述如表3.2所示。

图3.1　车灯控制系统软件架构设计

表3.2　车灯控制系统软件组件

软件组件	类型	描述	所属 ECU
SensorSWC(2)	Sensor/Actuator SWC	处理光强传感器信号	SensorECU
LightRequestSWC(5)	Application SWC	管理车灯控制指令、CAN 通信指令	LightECU
LightControlSWC(7)	Application SWC	控制并判断车灯状态	LightECU
EcuBaseSWC(1)	Application SWC	管理 ECU 相关基本状态	LightECU
EcuAliveIndicatorSWC(6)	Application SWC	设置单片机调试指示灯状态	LightECU
IOAbstractionSWC(3)	ECU Abstraction SWC	实现 I/O 硬件抽象层	LightECU
DisplaySWC(4)	Application SWC	显示车灯状态	DisplayECU

（3）目标ECU软件组件设计

本书示例的目标ECU为LightECU，它主要包括5个软件组件，其中LightRequestSWC和LightControlSWC是实现车灯控制的关键软件组件，下面对这两个软件组件的功能进行具体介绍。

LightRequestSWC软件组件对于A型和B型车灯控制器共有的主要任务是负责检测并判断车灯开关的状态，对于B型车灯还负责接收并判断

外界光强情况；并对外输出车灯控制指令，对于A型车灯控制指令为0/1，即开与关；对于B型车灯控制指令为PWM占空比值，分为0、50%、100%。此外，当车灯开关关闭时间长于5分钟时，发出关闭LightECU　CAN通信的指令，一旦检测到车灯开关打开，则再恢复控制器CAN通信。

LightControlSWC软件组件主要有两个运行实体。RE_JudgeLightState通过采集车灯实际硬件上的控制信号，即对于A型车灯通过ADC采集电压信号，对于B型车灯则通过ICU采集PWM占空比，对采样值进行处理判断后通过运行实体间变量IRVJudgeLightState将车灯状态传递给RE_LightControl。RE_LightControl则负责将车灯状态与车灯类型通过端口发出，并且将车灯实际控制量传递给I/O抽象软件组件。

最终，LightECU中各软件组件端口接口/端口设计和LightECU中软件组件内部行为设计结果如表3.3及表3.4所示。

表3.3　LightECU中各软件组件端口接口/端口设计

软件组件	端口	端口接口	描述
LightRequestSWC	RPortGetButtonState RPortGetLightIntensity PPortControlLightCmd PPortNoComRequest	CS_IF_GetButtonState SR_IF_GetLightIntensity SR_IF_ControlLightCmd SR_IF_NoComRequest	请求获取车灯开关状态 接收外界光强信息 发出车灯控制指令 发出关闭 CAN 通信请求
LightControlSWC	RPortGetLightState RPortControlLightCmd RPortSetLightState PPortSendLightType PPortSendLightState	CS_IF_GetLightState SR_IF_ControlLightCmd CS_IF_SetLightState SR_IF_SendLightState SR_IF_SendLightState	请求获取车灯状态 接收车灯控制指令 请求设置车灯状态 发送车灯类型 发送车灯状态
EcuBaseSWC	RPortNoComRequest	SR_IF_NoComRequest	接收关闭 CAN 通信请求
EcuAliveIndicatorSWC	RPortEcuAliveIndication	CS_IF_EcuAliveIndication	请求设置指示灯亮灭(调试用)
IOAbstractionSWC	PPortGetButtonState PPortSetLightState PPortGetLightState PPortEcuAliveIndication	CS_IF_GetButtonState CS_IF_SetLightState CS_IF_GetLightState CS_IF_EcuAliveIndication	获取车灯开关状态 设置车灯状态 获取车灯状态 设置指示灯亮灭(调试用)

表3.4　LightECU中软件组件内部行为设计

软件组件	运行实体	描述	RTE 事件
LightRequestSWC	RE_LightRequest LightRequest_Init	管理车灯控制指令、CAN 通信指令 初始化	Timing Event(10ms) Init Event
LightControlSWC	RE_JudgeLightState RE_LightControl LightControl _Init	判断车灯当前状态 控制车灯状态 初始化	Timing Event(20ms) Data Received Event Init Event
EcuBaseSWC	RE_EcuBase_SWC	管理并请求 ECU 相关状态切换	Timing Event(10ms)
EcuAliveIndicatorSWC	RE_SetEcuAlive	请求设置指示灯亮灭(调试用)	Timing Event
IOAbstractionSWC	RE_GetButtonState RE_SetLightState RE_GetLightState RE_EcuAliveIndicator	获取车灯开关状态 设置车灯状态 获取车灯状态 设置指示灯亮灭(调试用)	Operation Invoked Event Operation Invoked Event Operation Invoked Event Operation Invoked Event

3.1.4 示例系统AUTOSAR架构

根据上述软件组件架构设计，本书示例软件组件向各ECU分配示意如图3.2所示。之后，则需要针对LightECU进行ECU级开发。

图3.2　软件组件向各ECU分配示意

根据上述设计，这里着重分析一下A型和B型车灯控制器软件在AUTOSAR架构下的异同。A型车灯控制器软件架构及信号流示意如图3.3所示，B型车灯控制器软件架构及信号流示意如图3.4所示。

图3.3　A型车灯控制器软件架构及信号流示意

图3.4　B型车灯控制器软件架构及信号流示意

可见，由于AUTOSAR高度的分层架构以及明确的接口定义，使得需求变更时，整个软件架构的修改量大大下降，软件模块复用度大大提升。对于A型和B型车灯控制器而言，除了应用层软件组件中相关控制算法需要进行修改外，主要需要更换MCAL中的相关模块以及修改IO硬件抽象层中一些MCAL接口函数的调用。具体而言，对于A型车灯，用DO直接控制车灯，并且用ADC采集车灯控制信号；对于B型车灯，通过PWM输出不同占空比的信号来控制车灯的亮度，并用ICU采集车灯控制信号。所以，需要在MCAL层根据需求配置不同的模块，但对于IO硬件抽象层，即IOAbstractionSWC软件组件的各运行实体名可以不变，仅改变内部的代码即可。例如：同样名为SetLightState的函数，在A型车灯中的实现是对DIO通道的操作；而在B型车灯中的实现则是对PWM通道的操作。从这点也折射出AUTOSAR的魅力所在！

3.2 ETAS AUTOSAR系统解决方案介绍

博世集团ETAS公司基于其强大的研发实力为用户提供了一套高效、可靠的AUTOSAR系统解决方案，该方案覆盖了软件架构设计、应用层模型设计、基础软件开发、软件虚拟验证等各个方面，如图3.5所示，其中深色部分为ETAS所提供的产品及服务。

图3.5　ETAS AUTOSAR系统解决方案

ETAS AUTOSAR相关的产品主要包括以下内容。

①ISOLAR-A：软件架构设计工具，支持整车级软件架构的设计，可用于符合AUTOSAR规范的汽车嵌入式系统软件开发中的系统级开发。

②ASCET：基于模型的AUTOSAR软件组件建模工具。

③RTA系列：由RTA-RTE、RTA-BSW、RTA-OS组成，可用于AUTOSAR ECU级开发，即RTE与BSW的配置及代码生成。

④ISOLAR-EVE：虚拟ECU验证平台。

3.3　本书AUTOSAR系统解决方案介绍

本书示例将遵循AUTOSAR方法论来进行开发，所用的AUTOSAR解决方案如图3.6所示。

图3.6　本书AUTOSAR系统解决方案

首先，使用Matlab/Simulink来实现部分软件组件级的开发，主要包括LightRequestSWC和LightControlSWC，并自动生成应用层软件组件代码及arxml描述文件，其中软件组件arxml描述文件作为AUTOSAR系统级开发的输入文件之一。

其次，使用ETAS ISOLAR-A工具来进行AUTOSAR系统的设计与配置，过程中会利用ISOLAR-A工具设计一些附加的SWC，主要包括

EcuAliveIndicatorSWC、EcuBaseSWC以及I/O硬件抽象层SWC。系统级开发最后会抽取出待配置ECU的信息，即LightECU的信息，进而可以进入ECU级开发阶段。

在ECU级开发阶段，基于ETAS　RTA系列工具（RTA-RTE、RTA-BSW、RTA-OS）来实现ECU级的开发，即RTE及除MCAL以外的BSW模块配置和代码生成；使用NXP MCAL配置工具来实现MCAL模块的配置及代码生成。

最后，进行代码集成，使用Wind　River编译器进行代码编译链接，生成单片机可执行的文件，并通过Lauterbach调试器将单片机可执行的文件烧写到MPC5744P开发板进行代码调试。

3.4 本章小结

本章首先介绍了本书中示例的开发需求，并在介绍ETAS AUTOSAR系统解决方案的基础上，介绍了本书中所用的AUTOSAR系统解决方案。本章是后续章节的一个总领，通过本章的学习，可以明确本书示例的开发需求，了解示例开发所用的AUTOSAR系统解决方案。这里需要指出，后续章节将主要使用本章所提的解决方案，围绕这个示例进行开发过程详解，但又不拘泥于此，以全面解析AUTOSAR方法论的具体实施为宗旨。

第4章　AUTOSAR软件组件级设计与开发

本章中，AUTOSAR软件组件级设计与开发主要是针对应用层软件。根据本书示例所用的AUTOSAR系统解决方案，需要先基于Matlab/Simulink进行应用层软件组件的模型建立，并配置生成符合AUTOSAR规范的代码及arxml描述文件，这是一种“自下而上”的工作流程。此外，还可以基于一种“自上而下”的工作流程来进行应用层软件组体的开发。下面介绍上述两种开发流程的具体实现方法。

4.1 Matlab/Simulink与Embedded Coder工具简介

4.1.1 Matlab/Simulink工具简介

Simulink是Matlab最重要的组件之一，它提供了一个动态系统建模、仿真和综合分析的集成环境。在该环境中，无须大量编写程序，只需要通过简单直观的鼠标操作就可以构造出复杂的系统。Simulink具有适应面广、结构和流程清晰以及仿真精细、贴近实际、效率高、灵活等优点。

Stateflow提供了一个编辑器和一些用于绘制状态机和流程图的图形对象。通过选择状态、转移和结点，然后将其拖入Stateflow编辑器，即可构建状态机。

4.1.2 Embedded Coder工具简介

Embedded Coder工具可以生成可读、紧凑且快速的C和C++代码，以便用于嵌入式处理器、目标系统快速原型板和量产中使用的微处理器。Embedded Coder工具丰富了Matlab Coder和 Simulink Coder的配置选项，并对其进行高级优化，从而可对生成代码的函数、文件和数据进行细粒度控制。这些优化可以提高代码执行效率，并有助于和已有代码、数据类型及标定参数进行集成。Embedded Coder可生成符合AUTOSAR和ASAP2软件标准的代码与描述文件。此外，它还可以提供可溯源性报告、代码接口文档和自动化软件验证，从而便于用户遵循DO-178、IEC 61508和ISO 26262等标准进行软件开发。

基于上述工具可以实现基于模型的设计（Model-Based Design，

MBD），其具有如下优势：

①图形化设计；

②早期验证；

③代码自动生成；

④文档自动化等。

4.2 基于Matlab/Simulink的软件组件开发

基于Matlab/Simulink的软件组件开发主要就是对AUTOSAR软件组件内部行为的实现，即实现内部算法。

4.2.1 Matlab/Simulink与AUTOSAR基本概念的对应关系

如前所述，Embedded Coder可基于Matlab/Simulink模型生成符合AUTOSAR规范的代码及描述文件，但在建模过程中需要按照一定的对应关系去设计AUTOSAR软件组件的各个组成元素，所以在使用Matlab/Simulink进行基于模型的设计前首先需要理清AUTOSAR与Matlab/Simulink元素的对应关系，常用的一些元素对应关系如表4.1所示。

表4.1 AUTOSAR与Matlab/Simulink元素的对应关系

AUTOSAR 概念	Matlab/Simulink 概念
Composition(部件)	VirtualSubsystem(虚拟子系统)
Atomic Software Component(原子软件组件)	Virtual/Non-virtual Subsystem、Model(虚拟/非虚拟子系统、模型)
Sensor/Actuator Software Component(传感器/执行器软件组件)	VirtualSubsystem(虚拟子系统)
Runnable Entity(运行实体)	Function Call Subsystem(函数调用子系统)
RTE Event(RTE 事件)	Function Call(函数调用)
Port Interface(端口接口)	无
S/R PPort(S/R 供型端口)	Outport(输出端口)
S/R RPort(S/R 需型端口)	Inport(输入端口)
C/S PPort(C/S 供型端口)	Simulink Function(Simulink 函数模块)
C/S RPort(C/S 需型端口)	Function Caller(函数调用模块)
Inter Runnable Variable(运行实体间变量)	Signal(信号)

AUTOSAR与Matlab/Simulink数据类型的对应关系如表4.2所示。

表4.2　AUTOSAR与Matlab/Simulink数据类型的对应关系

AUTOSAR 数据类型	Matlab/Simulink 数据类型
boolean	boolean
float32	single
float64	double
sint8	int8
sint16	int16
sint32	int32
uint8	uint8
uint16	uint16
uint32	uint32

4.2.2　软件组件内部行为建模方法

在Matlab/Simulink中，可以直接进行软件组件内部行为设计，对照前述Matlab/Simulink与AUTOSAR基本概念的对应关系可知，Simulink中利用Function Call Subsystem（函数调用子系统）来表征AUTOSAR软件组件的Runnable Entity（运行实体）；利用Function Call（函数调用）来表征AUTOSAR软件组件的RTE Event（RTE事件）。基于这一思路，这里以LightControlSWC软件组件为例进行内部行为建模方法及注意点介绍。

首先，需要新建一个函数调用子系统，如图4.1所示。Function Call Subsystem表示一个运行实体，其内部模型为运行实体所封装的算法，而function（）就是运行实体的RTE事件。

图4.1　Function Call Subsystem创建

遵循上述思路，结合LightControlSWC软件组件的功能需求，可以完成它的内部行为建模，如图4.2所示。

图4.2　LightControlSWC软件组件Simulink模型

其中，由于Inport 1（Event_JudgeLightState）和Inport 2（Event_LightControl）作为RTE事件，所以需要勾选Output function call。并且，对于周期性触发的RE_JudgeLightState，作为RTE事件的Inport 1中的Sample time则对应该运行实体的调用周期，如图4.3所示。

图4.3　Event_JudgeLightState和Event_LightControl Inport配置

其次，由于LightControlSWC软件组件中的两个运行实体间涉及运行实体间变量（IRV），所以在建模过程中需要对相关信号进行标示，这样才能在后续模型配置过程中被识别为Inter-Runnable Variables。例如图4.2中的IRVJudgeLightState。

同理，可以开发出LightRequestSWC软件组件的Simulink模型，如图4.4所示。

图4.4　LightRequestSWC软件组件Simulink模型

4.2.3　AUTOSAR客户端/服务器机制的实现方法

在Matlab/Simulink中可以完成AUTOSAR客户端/服务器机制的实现。由于客户端/服务器机制的本质是函数调用关系，即客户端调用服务器的函数。根据AUTOSAR与Matlab/Simulink元素对应关系表中C/S PPort（C/S供型端口），即Server端，对应Simulink Function（Simulink函数模块）；C/S RPort（C/S需型端口），即Client端，对应Function Caller（函数调用模块）。

本书示例中，与IOAbstractionSWC软件组件交互的应用层软件组件需要用到C/S通信，这里以LightControlSWC软件组件中的RE_JudgeLightState运行实体中的Client端设计为例进行讲解。该端口需

要调用IOAbstractionSWC软件组件中的函数，并从中读取灯的状态信息，对于A型车灯而言是读取A/D转换值，对于B型车灯而言是读取ICU捕获到的脉冲占空比值。在Simulink中的实现如图4.5和图4.6所示。

图4.5　RE_JudgeLightState运行实体中Client端实现（一）

图4.6　RE_JudgeLightState运行实体中Client端实现（二）

在Function Caller配置中，Function prototype为C/S接口的操作（Operation）名，可通过配置Input argument specficaitions和Output argument specficaitions来进行Data Element定义。

4.3　软件组件代码及描述文件配置生成

虽然通过Matlab/Simulink模型可以直接生成符合AUTOSAR规范的代码与软件组件arxml描述文件，但在生成之前需要对软件组件相关的信息进行配置，如端口接口、端口、运行实体等，并且需要和模型中的元素进行映射。所以，开发者应该对AUTOSAR软件组件相关基础知识有一定认识，并且明确这些概念与Matlab/Simulink模型中元素的对应关系。下面详细讲解通过Matlab/Simulink模型直接生成符合AUTOSAR规范的代码与软件组件arxml描述文件的方法，并对自动生成的文件进行解释说明。

4.3.1　求解器及代码生成相关属性配置

在进行代码配置生成之前，首先要保证当前求解器（Solver）所选取的步长模式是定步长（Fixed-step）的，即Solver　options选为Fixed-step模式，如图4.7所示。

图4.7　Solver模式配置

其次，要配置系统目标文件，在Simulink主菜单中点击Code，选择C/C++ Code中的Code Generation Options选项，在弹出的界面中选择Solver配置，把System target file更改为autosar.tlc，如图4.8所示。

图4.8　Code Generation配置

其中，tlc是target language compiler（目标语言编辑器）的全称，其类似脚本语言，可以控制代码生成的格式。该文件默认存放在如下路径：Matlab安装目录\toolbox\rtw\targets\AUTOSAR\AUTOSAR\AUTOSAR.tlc。

此时，在Code Generation根目录下会出现AUTOSAR Code Generation的配置选项，如图4.9所示。

图4.9　AUTOSAR Code Generation配置

①Generate XML file for schema version选项中可以选择生成描述文件的AUTOSAR版本，从2.1版本开始到4.2版本，本书中选择生成4.2版本的arxml描述性文件。

②Maxium SHORT-NAME length属性可以设置命名的最大长度，即之后在配置软件组件相关特性时自定义名字的最大长度。由于AUTOSAR中对所有命名有以下规则：对于名称识别符，最多可包含32个字符，以字母开头，包含字母、数字和下划线；对于路径识别符至少含有一个“/”字符，分隔符之间的字符串最多包含32个字符，以字符开头，包含字母、数字和下划线。

③Use AUTOSAR compiler abstraction macros选项则是开启或者关闭AUTOSAR规范中所定义的一些宏，如FUNC等。

4.3.2 模型配置

当在Matlab/Simulink中设计完软件组件内部行为，并且完成了上述准备工作后，可以进行模型的配置，即将模型配置成AUTOSAR软件组件。在Simulink主菜单中点击Code，选择C/C++ Code中的Configure Model as AUTOSAR Component选项，如图4.10所示。

图4.10 AUTOSAR Code Generation配置

此时，会弹出如图4.11所示界面，该界面主要包括两大部分：Simulink-AUTOSAR Mapping与AUTOSAR Properties，可通过左下角选择按钮进行切换。

图4.11　AUTOSAR Code Generation配置（Simulink-AUTOSAR Mapping）

其中，Simulink-AUTOSAR　Mapping主要可以将Simulink中所建模型的元素与AUTOSAR软件组件相关元素进行对应，这里可以参考Matlab/Simulink与AUTOSAR基本概念的对应关系部分所介绍的内容。在Simulink　Mapping菜单下可见模型的名字，如图4.11所示是LightControlSWC的模型，这里默认将其映射到了一个同名的软件组件。在其下面罗列了一些Simulink模型中的元素，有以下几种，它们均可与AUTOSAR软件组件相关元素进行映射。

①Inports：输入端口。

②Outports：输出端口。

③Entry Point Functions：入口点函数。

④Function Callers：函数调用模块。

⑤Data Transfers：数据传递。

当切换到AUTOSAR　Properties配置界面时，可以配置与AUTOSAR软件组件相关的元素以及描述文件生成选项，可配置的内容主要如下，

如图4.12所示。

图4.12　AUTOSAR Code Generation配置（AUTOSAR Properties）

（1）原子软件组件（AtomicComponents）配置

①ReceiverPorts（接收者端口）；

②SenderPorts（发送者端口）；

③SenderReceiverPorts（发送者接收者端口）；

④ModeReceiverPorts（模式接收者端口）；

⑤ClientPorts（客户端端口）；

⑥ServerPorts（服务器端口）；

⑦NvReceiverPorts（非易失性数据接收者端口）；

⑧NvSenderPorts（非易失性数据发送者端口）；

⑨NvSenderReceiverPorts（非易失性数据发送者接收者端口）；

⑩Runnables（运行实体）；

⑪IRV（运行实体间变量）。

（2）端口接口（Port Interfaces）配置

①S-R Interfaces（发送者-接收者接口）；

②M-S Interfaces（模式转换接口）；

③C-S Interfaces（客户端-服务器接口）；

④NV Interfaces（非易失性数据接口）。

（3）计算方法（CompuMethods）配置

（4）XML文件选项（XML Options）配置

下面以LightControlSWC模型的配置过程为例，详细介绍将Simulink模型配置成AUTOSAR软件组件的具体方法。由于Simulink-AUTOSAR Mapping界面是将AUTOSAR软件组件元素与Simulink元素进行映射，所以建议先在AUTOSAR Properties界面完成软件组件元素的定义与相关配置，再切换到Simulink-AUTOSAR Mapping界面完成AUTOSAR软件组件元素与Simulink元素的映射。

4.3.3 AUTOSAR Properties配置

如前所述，由于端口接口是对端口属性的描述，所以先配置端口接口，根据LightControlSWC软件组件设计的需求可知，它涉及S-R Interfaces和C-S Interfaces。

对于S-R Interfaces，可以点击“+”进行新建，给新建的接口赋予名字后，可以点击每个新建的S-R Interfaces，完成数据元素（DataElements）的添加，如图4.13所示。

图4.13　S-R Interfaces配置

对于C-S Interfaces，可以点击“+”进行新建，给新建的接口赋予名字后，可以点击每个新建的C-S Interfaces，完成操作（Operations）和参数（Arguments）的添加，如图4.14所示。

图4.14　C-S Interfaces配置

在定义完LightControlSWC软件组件所需要的端口接口之后，可以配置原子软件组件（AtomicComponents）中的内容，即完成对软件组件LightControlSWC的设计。根据需求，LightControlSWC软件组件主要涉及ReceiverPorts、SenderPorts、ClientPorts、Runnables和IRV，这些也是开发过程中较常用的元素。

对于所有的Port配置而言，总体上都是先新建一个Port，再引用一个上述定义的端口接口。如图4.15所示为LightControlSWC原子软件组件的ReceiverPorts、SenderPorts、ClientPorts的配置。在SenderPorts中，这里将两个不同的端口引用了同一个端口接口，对于最终的数据元素映射关系将在Simulink-AUTOSAR Mapping过程中完成。

图4.15　原子软件组件Port相关配置

之后，则需要定义运行实体及其RTE事件。可以点击“+”进行新建，给新建的运行实体赋予名字后，在Event界面可以点击Add　Event新建一个RTE事件。例如，按需求可知，LightControlSWC软件组件包含两个运行实体，其中一个是周期性触发的；另一个则是收到端口数据后触发的。

如图4.16所示是周期性触发的运行实体设计过程，添加一个TimingEvent即可。

图4.16　周期性触发运行实体配置

而对于收到端口数据后触发的运行实体，可以选择DataReceivedEvent，并且需要选择一个触发源，即选择一个端口数据元素，如图4.17所示。

图4.17　收到端口数据后触发的运行实体配置

Runnable_Init运行实体是Simulink自动创建的，用于完成Simulink模型中的初始化操作，一般为单次运行，其配置如图4.18所示。注意：AUTOSAR 4.0以上版本才支持InitEvent的配置。

图4.18　初始化运行实体配置

软件组件配置的最后是IRV，即运行实体间变量的配置。LightControlSWC软件组件涉及一个运行实体间变量，点击“+”添加即可，如图4.19所示。

图4.19　IRV配置

最后，是XML Options（XML文件选项）配置，这里建议Export XML file packaging选择Single file，这样描述文件基本可以集中生成在一个arxml中，便于之后的操作，如图4.20所示。

图4.20　XML Options配置

4.3.4　Simulink-AUTOSAR Mapping配置

完成了AUTOSAR Properties配置后，则需要进行Simulink-AUTOSAR Mapping配置，即将Simulink模型中的元素与AUTOSAR软件组件元素映射。

（1）Inports/Outports

Simulink模型中的Inports和Outports需要与刚才所定义的AUTOSAR软件组件的端口映射，并且需要选择端口的数据元素，这尤其体现在当多个端口引用同一个端口接口，并且该端口接口中定义了多个数据元素

的情形，如图4.21中Inports/Outports配置所示。DataAccessMode可定义数据访问模式，这里采用隐式（Implicit）模式。

图4.21　Inports/Outports配置

（2）Entry Point Functions

入口点函数（Entry Point Functions）主要是配置模型中的函数与AUTOSAR软件组件运行实体的映射关系。如前所述，每个运行实体本质上就是一个函数，Simulink中若采用函数调用子系统（Function Call Subsystem），在映射时就能将其看作一个运行实体。LightControlSWC软件组件中各运行实体与模型元素的映射配置结果如图4.22所示。

图4.22　Entry Point Functions配置

（3）Function Callers

函数调用（Function Callers）主要是配置Simulink模型元素与Client端口的映射，这也从一个侧面反映了Client/Server模式的本质是函数的调用关系；其中，函数就是Client-Server Interface中所定义的操作（Operation），有了这个概念就可以完成Function Callers中的配置。LightControlSWC软件组件中各Client端口与模型元素的映射结果如图4.23所示。

图4.23 Function Callers配置

（4）Data Transfers

数据传递（Data Transfers）指的就是两个函数调用子系统间传递的数据，由于函数调用子系统对应运行实体；所以，此处各组数据传递关系需要和AUTOSAR软件组件中的运行实体间变量（IRV）进行映射。LightControlSWC软件组件中各运行实体间变量与模型元素的映射结果如图4.24所示。

图4.24 Data Transfers配置

4.3.5 符合AUTOSAR规范的代码及描述文件生成

上面已经详细讲述了在Matlab/Simulink中进行软件组件设计开发的方法，主要分为两阶段。

①第一阶段：Simulink建模阶段。必须遵循AUTOSAR软件组件概念与Simulink模型概念的对应关系进行模型设计，为后续配置做准备。

②第二阶段：模型配置阶段。进行AUTOSAR Properties配置和Simulink-AUTOSAR Mapping配置，即完成AUTOSAR软件组件元素的设计及其与Simulink模型元素的映射。

当完成上述两阶段的工作后，就可进行符合AUTOSAR规范的代码及描述文件生成。在Simulink主菜单中点击Code，选择C/C++ Code中的Build Model选项即可生成，如图4.25所示。最终，生成的文件会在Current Folder下。

图4.25　符合AUTOSAR规范的代码及描述文件生成方法

生成完成后，即可得到符合AUTOSAR规范的代码及描述文件。这里还是以LightControlSWC软件组件生成结果为例进行分析。最终，该软件组件一共生成11个文件，其中各文件的主要内容与作用描述如下。

①LightControlSWC.c：包含所有软件组件运行实体的模型算法实现代码，每个运行实体分别对应一个函数。例如：

```
FUNC（void，LIGHTCONTROLSWC_CODE）RE_JudgeLightState
（void）
```

```
{
   uint8_T rtb_FunctionCaller;

   /*RootInportFunctionCallGenerator: '<Root>/RootFcnCall_InsertedFor_Event_JudgeLightState_at_outport_1' incorporates:
      *  SubSystem: '<Root>/RE_JudgeLightState'
   */
   /* FunctionCaller: '<S1>/Function Caller' */
   Rte_Call_RPortGetLightState_OPGetLightState（&rtb_FunctionCaller）；

   /* SignalConversion: '<S1>/TmpSignal ConversionAtIRVJudgeLightStateInport1' incorporates:
      *  DataTypeConversion: '<S1>/Data Type Conversion1'
      *  Switch: '<S1>/Switch'
   */
   Rte_Irv_Write_RE_JudgeLightState_IRVJudgeLightState（（uint8_T）（rtb_FunctionCaller > 4））；

   /* End of Outputs for RootInportFunctionCallGenerator: '<Root>/RootFcnCall_InsertedFor_Event_JudgeLightState_at_outport_1' */
}
```

②LightControlSWC.h：包含系统的数据结构、函数外部声明。

③LightControlSWC_private.h：包含软件组件私有函数的定义和数

据声明。

④LightControlSWC_types.h：包含用typedef定义的模型中所有的参数结构体。

⑤rtwtypes.h：定义Matlab/Simulink的数据类型格式。

⑥LightControlSWC.arxml：这个arxml描述性文件里描述了LightControlSWC软件组件的端口、端口接口、数据类型和内部行为等AUTOSAR软件组件中的元素。

⑦LightControlSWC_external_interface.arxml：这个arxml描述性文件里描述了LightControlSWC 软件组件所包括的C/S接口信息。

⑧其他：主要都是一些RTE相关的定义文件。

4.4　在Simulink中导入软件组件描述文件——“自上而下”的工作流程

“自上而下”的工作流程有别于“自下而上”的工作流程，其需要先在AAT（AUTOSAR Authoring Tool）工具（如ISOLAR-A）中完成软件组件框架设计，并将软件组件arxml描述文件导入Matlab/Simulink完成内部算法的实现，然后再通过Matlab/Simulink生成符合AUTOSAR规范的代码及arxml描述文件。这里以用于调试的EcuAliveIndicatorSWC软件组件为例进行“自上而下”工作流程的讲解。

这里假设已经基于ISOLAR-A工具完成了EcuAliveIndicatorSWC软件组件框架的设计，并得到了其描述文件，在Matlab中编写如下脚本文件：点击运行Run即可完成描述文件导入及模型创建，如图4.26所示；之后，Matlab/Simulink会根据软件组件描述文件中的信息自动生成软件组件模型，如图4.27所示，这样就可以基于软件组件模型框架进行内部算法的实现；最终，再生成符合AUTOSAR规范的代码及描述文件即可。

图4.26　软件组件描述文件导入与模型创建脚本运行

图4.27　EcuAliveIndicatorSWC软件组件描述文件导入及模型创建结果

```
obj = arxml.importer（{'EcuAliveIndicatorSWC.arxml', 'I
SOLAR_PlatformTypes.arxml'}）
createComponentAsModel（obj, '/IoHwAbstraction/EcuAlive
IndicatorSWC', 'CreateInternalBehavior', true）
```

4.5 本章小结

本章在介绍Matlab/Simulink与Embedded Coder工具的基础上，对基于Matlab/Simulink进行AUTOSAR软件组件开发并生成符合AUTOSAR规范的代码及arxml描述文件的方法（即“自下而上”的工作流程）进行了较为全面的讲解；然后，还对基于“自上而下”工作流程的开发方法进行了介绍。通过本章的学习，可以对AUTOSAR软件组件有一个较为感性的认识，并学会基于Matlab/Simulink进行AUTOSAR软件组件开发的基本方法。

第5章　AUTOSAR系统级设计与配置

如前所述，开发者可以先在系统级设计工具ISOLAR-A中设计软件组件框架，包括端口接口、端口等，即创建各软件组件arxml描述性文件；再将这些软件组件描述性文件导入到行为建模工具，如Matlab/Simulink中完成内部行为建模。亦可以先在行为建模工具中完成逻辑建模，再把生成的描述性文件导入到系统级设计工具中完成系统级设计与配置。

根据本书所使用的AUTOSAR系统解决方案，先前已在Matlab/Simulink中完成了部分应用层软件组件的建模，并生成了符合AUTOSAR规范的代码与arxml描述性文件，此处可以直接导入ISOLAR-A系统级设计工具，进行后续配置。

5.1 ETAS ISOLAR-A工具简介

ISOLAR-A工具是ETAS公司专门为符合AUTOSAR规范的汽车嵌入式系统软件开发所推出的工具，如图5.1所示。众所周知，在开发AUTOSAR系统时，软件架构设计是至关重要的。得益于博世集团在ECU开发过程中的技术沉淀及其与全球OEM的密切合作，ISOLAR-A工具是一款高效的AUTOSAR软件架构设计工具。

图5.1 ETAS ISOLAR-A工具

ISOLAR-A工具是基于Eclipse平台开发的，它具有开放的、可扩展的特点，这为集成用户的特定功能提供了可能性。目前，ISOLAR-A 支持AUTOSAR R4.0、R4.1、R4.2（包括R4.2.2）和R4.3。

ISOLAR-A工具主要具有以下功能与特色：

①可以进行整车级别的软件组件架构设计；

②设计符合AUTOSAR规范的软件组件，包括软件组件端口、端口接口、运行实体等的设计，还可以导入/导出软件组件arxml描述文件，并支持与其他基于模型的设计（Model Based Design，MBD）开发工具协同开发，如ETAS ASCET、Matlab/Simulink，它们之间可以通过arxml描述文件进行软件组件信息交互；

③导入传统格式的文件，如DBC（包括SAE J1939和ISOBUS 11783）、FIBEX、LDF（包括SAE J2602）和ODX等，自动化地完成这些大工作量的任务，不仅可以提高开发效率，而且还可以减少开发错误；

④可以进行AUTOSAR系统级设计，并可为单核或多核ECU完成系统以及ECU信息抽取工作；

⑤可以集成AUTOSAR系统和应用软件；

⑥采用标准的AUTOSAR XML文件格式（arxml），很容易连接到SVN等软件配置和版本管理系统等。

作为ETAS整体AUTOSAR解决方案的一部分，ISOLAR-A可与ETAS和其他供应商的AUTOSAR工具进行集成。特别是ETAS的RTA-RTE（AUTOSAR运行环境生成器）和RTA-OS（操作系统），还可以集成AUTOSAR-R4.x的基础软件，如ETAS RTA-BSW。

5.2 ETAS ISOLAR-A工具入门

5.2.1 ISOLAR-A安装方法

ISOLAR-A工具安装方法较为便捷，按照安装提示默认操作即可。

双击打开ETAS ISOLAR-A的安装包文件夹，双击运行Autostart.exe，会出现如图5.2所示的安装界面，然后点击Main，会弹出如图5.3所示界面，界面中提示本安装包仅支持装有64位操作系统的计算机。

图5.2 ETAS ISOLAR-A工具安装步骤（一）

图5.3　ETAS ISOLAR-A工具安装步骤（二）

点击ISOLAR-A Product Installation（for 64bit OS only），进入正式安装界面，如图5.4所示。

图5.4　ETAS ISOLAR-A工具安装步骤（三）

点击下一步Next，会弹出如图5.5所示的liscense管理条约，用户在仔细阅读后，可以勾选

I read and accept the terms in the liscense aggrement，之后点击下一步Next。

图5.5　ETAS ISOLAR-A工具安装步骤（四）

点击下一步后，会弹出如图5.6所示界面，用户可以选择ISOLAR-A的安装路径。在文件夹命名时可以附加一些软件的版本信息，如这里使用的是ISOLAR-A　9.2.1，则安装文件夹可命名为ISOLAR_V921，这可以便于今后多版本共存情况下的管理。

图5.6　ETAS ISOLAR-A工具安装步骤（五）

在选择安装路径之后，可以点击下一步Next，会弹出如图5.7所示的窗口。这时会让用户勾选需要安装的程序，主要包括一些工具运行需要的Visual C++插件、ETAS Liscense Manager以及ISOLAR-A。初次安装建议全部勾选，便于后期使用。界面右下角会显示工具安装需要的空间以及计算机各盘的空间大小。最后，可以点击安装Install进行软件安装。

图5.7　ETAS ISOLAR-A工具安装步骤（六）

此时，会弹出如图5.8所示的界面，进度条显示安装状态。安装完毕后，会弹出如图5.9所示的界面，点击Finish即可。至此，ETAS ISOLAR-A工具安装过程结束。

图5.8　ETAS ISOLAR-A工具安装步骤（七）

图5.9　ETAS ISOLAR-A工具安装步骤（八）

5.2.2　ISOLAR-A界面说明

ISOLAR-A工具的界面十分友好，如图5.10所示，其主要分为主菜单、AR Explorer View、Description View、Properties View和Graphical Editors View。

图5.10　ETAS ISOLAR-A工具界面

其中，主菜单关键选项的功能在下面工具使用过程中将逐步涉及。由于AUTOSAR配置工具主要是针对arxml等格式的描述文件的操作，所以其可视为一个强大的XML文件编辑器。AR Explorer View、Description View、Properties View和Graphical Editors View的操作也是主要针对描述文件中的元素展开的。

AR Explorer View是工程文件浏览器，一般推荐选择Show Abstraction Groups Only（图5.11），这样ISOLAR-A工具将根据AUTOSAR中的相关概念将工程文件夹中所有描述文件中的元素进行分类，如Data Types、Interfaces、Components，从而便于开发。

图5.11　AR Explorer浏览模式选择

Filesystem Navigator是arxml浏览器，切换到该界面将显示出工程文

件夹所包含的arxml描述文件，如图5.12所示。

图5.12　arxml文件浏览器

Description View提供所选元素的文件名和所属于的AUTOSAR Package信息。此外，还显示AUTOSAR元素类型的描述。

Properties View可为选定的AUTOSAR容器提供相关配置。

Graphical Editors View则以图形和表格的形式更直观地、更清楚地展现AUTOSAR描述文件内容，并可以基于此配置相关的AUTOSAR元素。

5.3　基于ISOLAR-A的软件组件设计方法

ISOLAR-A作为系统级设计工具，可以进行软件组件的设计，主要包括：数据类型定义、端口接口设计、端口设计、软件组件设计、运行实体设计、运行实体间变量设计等。虽然，本书所用AUTOSAR解决方案中使用Matlab/Simulink进行应用层软件组件的设计，并直接生成了符合AUTOSAR规范的代码及描述文件，但这里还是以实例介绍一下ISOLAR-A中进行AUTOSAR软件组件设计的方法。

5.3.1　AUTOSAR工程创建

双击ISOLAR-A快捷方式，打开ISOLAR-A工具，会提示选择工作空间（Workspace），如图5.13所示。点击OK后，会进入ISOLAR-A主界面。

图5.13　ISOLAR-A工作空间选择

进入主界面后，点击File→New→AUTOSAR　Project新建工程，若

AR Explorer界面中没有任何旧工程，直接点击该界面中的New AUTOSAR Project新建工程即可。进入新建工程界面（图5.14），可以输入工程名字，选择AUTOSAR规范版本，勾选是否导入基础数据类型等。

图5.14　新建AUTOSAR Project

信息输入完成后，可以点击Finish。这时，AR Explorer界面中就会出现一个新建的工程，如图5.15所示。其中，与软件组件描述（Software Component Description）相关的元素都在Software文件夹中；与系统描述（System Description）相关的元素都在System文件夹中；与ECU描述（ECU Description）相关的元素都在Bsw文件夹中。

图5.15　AUTOSAR Project

5.3.2　数据类型定义

在介绍AUTOSAR软件组件基本概念的时，已经提到AUTOSAR定义了三种数据类型，即应用数据类型（Application Data Type，ADT）、实现数据类型（Implementation Data Type，IDT）、基本数据类型（Base Type）。这里介绍它们在工具中的定义及引用方法。由于数据类型属于软件组件相关元素，所以可以点开Software文件夹，如图5.16所示。

图5.16　Data Types文件夹

（1）数据类型创建

这里以Application Data Type创建方法为例介绍数据类型设计方法，其他两种数据类型类似。假设需要新建一个应用数据类型ButtonState，它代表开关的状态。它引用一种计算方法，0表示OFF；1表示ON。

右击Data Types→Create Data Types→Create Primitive Data Types→Elements|A
如图5.17所示。

图5.17　新建Application Data Type

此时，将弹出新建AUTOSAR元素的界面，需要输入元素名称；勾选是否新建package，package是arxml描述文件中的一个概念；若新建则需要输入package的名称，并选择需要放在哪个arxml文件中，可以双击工程文件夹，选择已有文件，也可以新建一个arxml文件专门存放Application Data Type。这里新建一个package，并且存放在Application Data Type.arxml文件中，如图5.18所示。最终，Application Data Type新建结果如图5.19所示。

图5.18　新建Application Data Type AUTOSAR Element

图5.19　Application Data Type新建结果

（2）计算方法型创建

和数据类型密切相关的是计算方法（Compu　Method），这里新建一个计算方法ButtonState来实现0表示OFF，1表示ON的对应关系。右键点击Software→Create　Compu　Methods→Elements|Compu　Method（图5.20），之后需要和先前新建数据类型一样新建一个AUTOSAR元素，方法不再重复。最终，Compute Method新建结果如图5.21所示。

图5.20　新建Compu Method

图5.21　Compu Method新建结果

双击刚才新建的计算方法ButtonState，将会弹出计算方法类型选择窗口（图5.22），其中较常用的计算方法的描述如下。

图5.22　Compu Method类型选择

①相同（IDENTICAL）：内部值与物理值是相同的，不需要进行转换。

②线性关系（LINEAR）：物理值是将内部值通过线性转换得到的，一般是内部值乘以一个比例因子（Factor）加上一个偏移量（OFFSET）得到物理值。

③有理函数（RAT_FUNC）：与线性转换类似，但转换仅限于有理函数。

④文本（TEXTTABLE）：将内部值转换为文本元素。

⑤比例线性和文本（SCALE_LINEAR_AND_　TEXTTABLE）：线性和文本相结合。

这里选择文本（TEXTTABLE）类型。

通过点击Add CompuScale可以添加TESTTABLE中的语句，来实现前述ButtonState的计算方法，如图5.23所示。

图5.23 Compu Method设计结果

设计完计算方法ButtonState后，就可以在先前建立的应用数据类型ButtonState中进行引用，如图5.24所示。

图5.24 Application Data Type设计结果

对于Application Data Type与Implementation Data Type的数据类型映射则需要在后续软件组件设计阶段完成，因为这种映射关系不是唯一

的，和实际需求密切相关。

5.3.3 端口接口设计

这里着重讲解常用的发送者-接收者接口（Sender-Receiver Interface，S/R）与客户端-服务器接口（Client-Server Interface，C/S）设计方法。

（1）为端口接口创建一个arxml描述文件

在定义AUTOSAR元素之前，先新建一个arxml描述文件专门存放端口接口定义描述。右键点击LightECU→New→AUTOSAR File（图5.25），之后会弹出如图5.26所示的界面，这里新建一个名为Interface.arxml的描述文件，并同时创建一个名为Interface的AR Package，点击Finish。最终，AUTOSAR File创建结果如图5.27所示。

图5.25　新建Interface AUTOSAR File 1

图5.26　新建Interface AUTOSAR File 2

图5.27　AUTOSAR File创建结果

（2）发送者-接收者类型接口设计

右键点击Software→Create Interfaces→Create Port Interfaces→Elements|Sender Recei 5.28）。这里也可以看出前面所提到的其他几种端口接口类型。之后会弹出如图5.29所示的界面，与先前定义数据类型时不同的是，由于已经创建了Interface.arxml文件，所以这里需要在下面的文件界面中选择它，并且选择Use Existing AR Package，选择名为Interface的AR Package Path，点击Finish。

图5.28　新建Sender-Receiver接口1

图5.29　新建Sender-Receiver接口2

双击刚才创建的SR_IF_SendLightState接口可进入Interface设计界面，点击Add VariableData Prototype可以添加接口的数据元素，在输入数据元素名字的同时，需要引用相应数据类型，这里均引用相应的应用

数据类型。最终，SR_IF_SendLightState接口设计结果如图5.30所示。

图5.30　Sender-Receiver接口设计结果

此时，在Interface.arxml文件中可以看到关于上述Sender-Receiver接口SR_IF_SendLightState的定义信息，其中主要的信息用粗体字标示。

```
    <SENDER-RECEIVER-INTERFACE>
  <SHORT-NAME>SR_IF_SendLightState</SHORT-NAME>
  <IS-SERVICE>false</IS-SERVICE>
  <DATA-ELEMENTS>
    <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>DESendLightType</SHORT-NAME>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE"
>/ApplicationDataTypes/LightType</TYPE-TREF>
    </VARIABLE-DATA-PROTOTYPE>
    <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>DESendLightState</SHORT-NAME>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE"
>/ApplicationDataTypes/LightState</TYPE-TREF>
    </VARIABLE-DATA-PROTOTYPE>
  </DATA-ELEMENTS>
```

```
</SENDER-RECEIVER-INTERFACE>
```

（3）客户端-服务器类型接口设计

设计完发送者-接收者类型接口后，可以直接切换至Client Server Interface设计界面，点击Add CS Interface创建一个C/S接口，名称为CS_IF_GetLightState；点击Add CS Operation添加一个操作，名称为OPGetLightState；点击Add ArgumentData Prototype添加参数DEGetLightState，由于是从底层读上来的数据，所以参数方向为OUT，并为其引用一个数据类型。客户端-服务器类型接口设计结果如图5.31所示。

图5.31　客户端-服务器类型接口设计结果

此时，在Interface.arxml文件中可以看到关于上述Client-Server接口CS_IF_GetLightState的定义信息，其中主要的信息用粗体字标示。

```
  <CLIENT-SERVER-INTERFACE>
<SHORT-NAME>CS_IF_GetLightState</SHORT-NAME>
<IS-SERVICE>false</IS-SERVICE>
<OPERATIONS>
  <CLIENT-SERVER-OPERATION>
    <SHORT-NAME>OPGetLightState</SHORT-NAME>
```

```
      <ARGUMENTS>
          <ARGUMENT-DATA-PROTOTYPE>
            <SHORT-NAME>DEGetLightState</SHORT-NAME>
            <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-
TYPE">/ApplicationDataTypes/LightState</TYPE-TREF>
            <DIRECTION>OUT</DIRECTION>
          </ARGUMENT-DATA-PROTOTYPE>
      </ARGUMENTS>
    </CLIENT-SERVER-OPERATION>
  </OPERATIONS>
</CLIENT-SERVER-INTERFACE>
```

5.3.4 软件组件设计

这里以建立LightControlSWC为例，详细介绍ISOLAR-A中软件组件的设计方法。

（1）软件组件新建

和上述端口接口设计一样，需要先为这个软件组件创建一个描述文件和AR Package，即右键点击LightECU→New→AUTOSAR File。这里创建一个名为LightControlSWC的arxml描述文件及名为LightControlSWC的AR Package（图5.32），点击Finish。

图5.32　新建LightControlSWC AUTOSAR File

之后，可以添加一个应用软件组件，右键点击Software→Create Components→Elements|Application Sw Component Type（图5.33），在弹出的界面中输入软件组件名，选择上面新建的LightControlSWC.arxml的描述文件及名为LightControlSWC的AR　Package，点击Finish。至此，一个软件组件就新建好了，可通过右

键点击软件组件LightControlSWC→Open With→Graphical Component Editor查看软件组件示意图，如图5.34所示。

图5.33　新建SWC

图5.34　LightControlSWC示意

（2）软件组件端口添加

双击软件组件LightControlSWC，可以打开Component Editor，点击Add　PPorts、RPorts、PRPorts会弹出如图5.35所示窗口，选择相应端口接口。为每个端口赋予名称即可，结果如图5.36所示。

图5.35　新建软件组件端口

图5.36　LightControlSWC端口设计结果

此时，在LightControlSWC.arxml描述文件中就会有上述端口的定义信息，下面选取PPortSendLightState与RPortGetLightState进行展示，其中一些重要信息用粗体字标示。

```
    <APPLICATION-SW-COMPONENT-TYPE>
  <SHORT-NAME>LightControlSWC</SHORT-NAME>
  <PORTS>
      <P-PORT-PROTOTYPE UUID="8bed18b6-b471-4de7-a10b-b
03b831fde56">
        <SHORT-NAME>PPortSendLightState</SHORT-NAME>
        <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-I
NTERFACE">/Interface/SR_IF_SendLightState</PROVIDED-
      INTERFACE-TREF>
      </P-PORT-PROTOTYPE>
      …
      <R-PORT-PROTOTYPE UUID="d5803fa8-ce7b-4e85-bf88-7
```

```
1796f319da2">
        <SHORT-NAME>RPortGetLightState</SHORT-NAME>
        <REQUIRED-INTERFACE-TREF DEST="CLIENT-SERVER-INT
ERFACE">/Interface/CS_IF_GetLightState</REQUIRED-
      INTERFACE-TREF>
      </R-PORT-PROTOTYPE>
      …
  </PORTS>
</APPLICATION-SW-COMPONENT-TYPE>
```

（3）软件组件内部行为设计

软件组件内部行为设计主要包括以下几点工作：

①创建软件组件内部行为（Internal Behaviour）；

②添加运行实体（Runnable）；

③添加运行实体与所属软件组件的端口访问（Port Access）；

④添加运行实的RTE发事件（RTE Event）；

⑤添加运行实体间变量（Inter Runnable Variable，IRV）；

⑥建立并添加数据类型映射（Data Type Mapping）。

第一步，创建软件组件内部行为（Internal Behaviour）。右键点击软件组件LightControlSWC→New Child→Internal Behaviours|Internal Behaviour（图5.37），将其命名为IBLightControlSWC。

图5.37　创建软件组件内部行为

第二步，添加运行实体（Runnable）。双击软件组件LightControlSWC，可以打开Component Editor，切换至Functions界面，点击Add Runnables可添加运行实体。为了便于读者全面认识软件组件运行实体设计过程，与前述Matlab/Simulink中一致，这里针对LightControlSWC软件组件也设计三个运行实体RE_JudgeLightState、RE_LightControl以及LightControl_Init，输入运行实体和函数名即可，结果如图5.38所示。

图5.38　LightControlSWC运行实体添加结果

第三步，添加运行实体与所属软件组件的端口访问（Port Access）。选中需要建立联系的软件组件，这里以RE_LightControl为例进行讲解。在右边的界面中可以添加Data Access Points与Server Call Points。

当切换到Data Access Points时，点击Add Access Points会弹出如图5.39所示的界面。这里可以选择数据的传输方式，有显示（Explicit）和隐式（Implicit）之分，区别在于采用Implicit模式，数据在RE调用之前读，在RE运行完成后写，这里使用Implicit模式。并且，从添加过程中也可以看出当两个不同端口引用同一个端口接口时，可以使用这一端口接口中所定义的一个或者多个数据元素，而并不一定是全部。Data Access Points添加结果如图5.40所示。

图5.39　Data Access Points添加界面

图5.40　Data Access Points添加结果

当切换到Server Call Points时，点击Add Server Call Points会弹出如图5.41所示的界面。客户端/服务器交互模式有同步（Synchronous）与异步（Asynchronous）之分，这里采用同步模式，即Client端一旦发出请求，就立即调用Server端的操作（函数）。Server Call Points添加结果如图5.42所示。

图5.41　Server Call Points添加界面

图5.42　Server Call Points添加结果

第四步，添加运行实体的RTE事件（RET　Event）。双击软件组件LightControlSWC，可以打开Component　Editor，切换至Events界面，点击Add　Events可显示如图5.43所示的列表，可见RTE事件种类非常多，比较常用的是Timing　Event（周期性事件）、Operation Invoked Event（操作调用事件）、Data Received Event（数据接收事件）、Init Event（初始化事件）等。

图5.43　RTE Event列表

根据需求，LightControlSWC软件组件中包含三个运行实体：RE_JudgeLightState、RE_LightControl以及LightControl_Init，它们的RTE事件类型选择也各不相同。

对于运行实体RE_JudgeLightState，其为周期性触发，周期为20ms。选择上述列表中的Timing　Event，将弹出如图5.44所示的窗口，

勾选待触发运行实体，填入触发周期，点击OK即可。

图5.44　Timing Event添加界面

对于运行实体RE_LightControl，当端口RPortControlLightCmd收到数据时触发，选择上述列表中的Data Received Event，将弹出如图5.45所示的窗口，勾选触发源端口数据元素，选择被触发的运行实体，点击OK即可。

图5.45　Data Received Event添加界面

LightControl_Init作为初始化运行实体，需要为其添加初始化事件，选择上述列表中的Init　Event即可，将弹出如图5.46所示的窗口，勾选被触发的运行实体，点击OK即可。

图5.46　Init Event添加界面

第五步，添加运行实体间变量（Inter Runnable Variable, IRV）。LightControlSWC软件组件中包含两个运行实体：RE_LightControl和RE_JudgeLightState，它们之间有运行实体间变量IRVJudgeLightState。双击软件组件LightControlSWC，可以打开Component Editor，切换至IRV界面，点击Add Impl IRV或Add Expl IRV，添加运行实体间变量，在右边添加两个运行实体表征读写关系即可，结果如图5.47所示。

图5.47　Inter Runnable Variable添加结果

第六步，建立并添加数据类型映射（Data Type Mapping），即Application Data Type与Implement Data Type的映射。右键点击Data Types→Create Data Types→Create Data Type Mapping Sets→Element 5.48），会弹出和先前创建AR Element一样的界面，选择一个arxml文件和一个AR Package即可，如图5.49所示。最终，Data Type Mapping设计结果如图5.50所示。

图5.48　新建Data Type Mapping 1

图5.49　新建Data Type Mapping 2

图5.50　Data Type Mapping设计结果

建立完Data Type Mapping后，可以将其添加到刚才建立的软件组件中。双击IBLightControlSWC，在References中的DataTypeMappings中可以添加刚才建立的数据类型映射DataTypeMappingSet，如图5.51所示。

图5.51　Data Type Mapping引用

至此，基于ISOLAR-A工具的软件组件设计已经全部完成，最终结果如图5.52与图5.53所示。

图5.52　LightControlSWC软件组件设计结果（一）

图5.53　LightControlSWC软件组件设计结果（二）

下面展示了软件组件描述文件的基本模板，其中囊括了一个完整的软件组件设计过程中所涉及的基本信息。

```
<APPLICATION-SW-COMPONENT-TYPE>
   <SHORT-NAME>...</SHORT-NAME>
```

```
   <PORTS>...</PORTS>
   <INTERNAL-BEHAVIORS>
   <INTERNAL-BEHAVIOR>
     <SHORT-NAME>...</SHORT-NAME>
     <DATA-TYPE-MAPPINGS>...</DATA-TYPE-MAPPINGS>
     <EVENTS>...</EVENTS>
     <INTER-RUNNABLE-VARIABLES>...</INTER-RUNNABLE-VARI
ABLES>
     <RUNNABLES>...</RUNNABLES>
    </INTERNAL-BEHAVIOR>
   </INTERNAL-BEHAVIORS>
</APPLICATION-SW-COMPONENT-TYPE>
```

端口定义的描述信息在先前已经展示过了，这里以LightControlSWC软件组件为例展示软件组件内部行为设计的描述。

数据类型映射关系的引用信息描述如下，其中的主要配置信息用粗体字标示。

```
    <DATA-TYPE-MAPPING-REFS>
  <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">/
DataTypeMapping/DataTypeMappingSet</DATA-TYPE-
        MAPPING-REF>
</DATA-TYPE-MAPPING-REFS>
```

运行实体的RTE事件的信息描述如下，其中的主要配置信息用粗体字标示。

```
    <EVENTS>
  <TIMING-EVENT UUID="e61160e0-1bb0-433c-8bf2-2516128e5
45f">
```

```
      <SHORT-NAME>TimingEvent_2ms</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-
           ENTITY">/LightControlSWC/LightControlSWC/I
BLightControlSWC/RE_JudgeLightState</START-ON-EVENT-
REF>
      <PERIOD>0.02</PERIOD>
  </TIMING-EVENT>
  <DATA-RECEIVED-EVENT UUID="cffa4af2-d48b-4737-8789-ae
13c745b351">
      <SHORT-NAME>DataReceivedEvent_ControlLightCmd</
SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-
           ENTITY">/LightControlSWC/LightControlSWC/I
BLightControlSWC/RE_LightControl</START-ON-EVENT-REF
>
      <DATA-IREF>
           <CONTEXT-R-PORT-REF DEST="R-PORT-
           PROTOTYPE">/LightControlSWC/LightControlSW
C/RPortControlLightCmd</CONTEXT-R-PORT-REF>
        <TARGET-DATA-ELEMENT-REF DEST="VARIABLE-DATA-
           PROTOTYPE">/Interface/SR_IF_ControlLightCm
d/DEControlLightCmd</TARGET-DATA-ELEMENT-REF>
      </DATA-IREF>
  </DATA-RECEIVED-EVENT>
   <INIT-EVENT UUID="1f074efc-6ee3-4886-a636-7e8c914307
ae">
      <SHORT-NAME>InitEvent</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-
```

```
        ENTITY">/LightControlSWC/LightControlSWC/IBL
ightControlSWC/LightControl_Init</START-ON-EVENT-REF
>
  </INIT-EVENT>
</EVENTS>
```

由于LightControlSWC软件组件涉及运行实体间变量，其相关信息描述如下，其中的主要配置信息用粗体字标示。

```
    <IMPLICIT-INTER-RUNNABLE-VARIABLES>
    <VARIABLE-DATA-PROTOTYPE UUID="e52213ae-3c97-4f9a-a
010-785854f1e9fb">
        <SHORT-NAME>IRVJudgeLightState</SHORT-NAME>
        <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE
">/ApplicationDataTypes/LightState</TYPE-TREF>
    </VARIABLE-DATA-PROTOTYPE>
</IMPLICIT-INTER-RUNNABLE-VARIABLES>
```

运行实体的信息描述如下，这里展现RE_JudgeLightState运行实体的描述文件，其中的主要配置信息用粗体字标示。

```
    <RUNNABLE-ENTITY UUID="4a158cff-8f14-484e-a975-d40a
08649c5d">
  <SHORT-NAME>RE_JudgeLightState</SHORT-NAME>
  <SERVER-CALL-POINTS>
      <SYNCHRONOUS-SERVER-CALL-POINT UUID="2cb36b00-087
2-4383-add6-963f990fddf2">
        <SHORT-NAME>SynchronousServerCallPoint_GetLigh
tState</SHORT-NAME>
```

```
        <OPERATION-IREF>
            <CONTEXT-R-PORT-REF DEST="R-PORT-
                PROTOTYPE">/LightControlSWC/LightContr
olSWC/RPortGetLightState</CONTEXT-R-PORT-REF>
            <TARGET-REQUIRED-OPERATION-REF DEST="CLIENT-
SERVER-
                OPERATION">/Interface/CS_IF_GetLightSt
ate/OPGetLightState</TARGET-REQUIRED-OPERATION-REF>
        </OPERATION-IREF>
      </SYNCHRONOUS-SERVER-CALL-POINT>
    </SERVER-CALL-POINTS>
    <SYMBOL>RE_JudgeLightState</SYMBOL>
        <WRITTEN-LOCAL-VARIABLES>
      <VARIABLE-ACCESS>
        <SHORT-NAME>WrittenLocalVariables_IRVJudgeLight
State</SHORT-NAME>
        <ACCESSED-VARIABLE>
            <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-
        PROTOTYPE">/LightControlSWC/LightControlSWC/IBL
ightControlSWC/IRVJudgeLightState</LOCAL-VARIABLE-RE
F>
        </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
    </WRITTEN-LOCAL-VARIABLES>
</RUNNABLE-ENTITY>
```

5.3.5 I/O硬件抽象层软件组件设计

虽然在AUTOSAR分层架构中提出了I/O硬件抽象层

（I/O Hardware Abstraction）的概念，但是由于其高度依赖于ECU的功能和设计，因此AUTOSAR没有标准化这部分软件的具体功能，AUTOSAR BSW软件供应商也就无法提供该部分的软件。所以，该部分的实现和一般SWC的定义类似，需要设计端口、端口接口等软件组件中所包含的元素，运行实体中的代码需要手动编写，从而来实现对微控制器抽象层（MCAL）相关接口的调用。本书示例涉及I/O硬件抽象层，所以需要在系统级设计之前，在ISOLAR-A中建立IOAbstractionSWC来对I/O硬件抽象层进行具体实施。

如前所述，I/O硬件抽象层实现是通过建立软件组件来实现的，所以本质上和先前建立的应用层软件组件一样，AUTOSAR为区分它们之间在概念上的区别，给它们赋予了不同的类别，ISOLAR-A中对各种类型的软件组件也展现了不同的图标，如图5.54所示。

图5.54　新建Ecu Abstraction SWC

此时，右键点击Software→Create Components→Elements|Ecu Abstraction Sw Component T和先前一样，需要定义一个AR Element，此处不再重复。

IOAbstractionSWC软件组件作为应用层软件组件与微控制器抽象层进行交互的桥梁，需要定义若干服务运行实体（Server Runnable），这样应用层软件组件通过引用Client/Server端口接口的端口作为Client就可

以调用Server端的操作，实现对MCAL接口的调用。

作为软件组件，在新建了软件组件框架后，需要先为IOAbstractionSWC软件组件定义端口。双击IOAbstractionSWC进入Component Editor，点击Add PPorts，将显示如图5.55所示的界面，选择四个ClientServerInterface类型的接口，即添加了四个作为Server端的端口。

图5.55　IOAbstractionSWC端口添加界面

添加完毕后，可以为四个端口命名。最终，IOAbstractionSWC端口添加结果如图5.56所示。

图5.56　IOAbstractionSWC端口添加结果

之后，需要设计IOAbstractionSWC的内部行为。由于IOAbstractionSWC需要实现对开关状态的读取、设置灯的状态、检测灯的状态、设置ECU调试指示灯的状态，所以这里设计四个运行实体，分别完成上述操作，结果如图5.57所示。

图5.57　IOAbstractionSWC运行实体设计结果

在添加了四个运行实体之后，需要为它们添加RTE事件，这里就需要使用到Operation　Invoked　Event来实现函数的调用触发关系，选择OperationInvokedEvent选项后（图5.58）将弹出如图5.59所示的界面，在勾选操作并选择对应运行实体后，点击OK即可，结果如图5.60所示。

图5.58　IOAbstractionSWC运行实体的RTE事件添加界面（一）

图5.59　IOAbstractionSWC运行实体的RTE事件添加界面（二）

图5.60　IOAbstractionSWC运行实体的RTE事件添加结果

至此，I/O硬件抽象层软件组件IOAbstractionSWC设计完成，设计

结果如图5.61与图5.62所示。

图5.61　IOAbstractionSWC软件组件设计结果（一）

图5.62　IOAbstractionSWC软件组件设计结果（二）

5.3.6　软件组件模板生成

在设计完各软件组件之后，可以生成软件组件模板，之后可以往各运行实体函数框架中填入相应算法。具体方法为右键点击待生成模板的软件组件，Generate→Component　Code-Frame，如图5.63所示，将弹出如图5.64所示的界面，点击Next，修改路径和文件名，点击Finish即可。

图5.63　软件组件模板生成

Component Code-Frame

Component Code-Frame creation wizard

Generates code templates with RTE API calls

Component Code-Frame options

Component Name: IOAbstractionSWC

(●) Single C-file Component Type

( ) Single C-file for each runnable selected of Component Type

[x] Use default templates

< Back | Next > | Finish | Cancel

Component Code-Frame

Single C File for Component type

Generates a single C file for all the Runnable Entities configured in the Component

Component Code-Frame options

Selected Component Name Type: IOAbstractionSWC

Source Folder: D:\ETASData\ISOLAR-A\workspace\LightECU\src Browse...

Component File Name: IOAbstractionSWC.c

< Back | Next > | Finish | Cancel

图5.64　软件组件模板生成界面

下面截取了IOAbstractionSWC软件组件中RE_SetLightState运行实体相关的函数框架代码。按照注释中的提示，在指定位置填入相应功能的实现代码即可。

```
    /* ***************************************************
*************
 * BEGIN: Banner
 *-------------------------------------------------------
-------------------------
 *                              ETAS GmbH
 *                   D-70469 Stuttgart, Borsigstr.14
 *-------------------------------------------------------
-------------------------
 *   Administrative Information（automatically filled
in by ISOLAR）
 *-------------------------------------------------------
-------------------------
 * Name: Code-Frame generation feature
 * Description:
 * Version: 9.0
 *-------------------------------------------------------
-------------------------
 * END: Banner
 ****************************************************
*********

 * Project: LightECU
 * Component: /IoHwAbstraction/IOAbstractionSWC
 * Runnable: All Runnables in SwComponent
```

```
 ***********************************************************
*********
 * Tool Version：ISOLAR-A V 9.0
 * Author：zhongwei
 * Date：星期二 二月 20 11：27：50 2018
 ***********************************************************
*********

#include "Rte_IOAbstractionSWC.h"

/*PROTECTED REGION ID（FileHeaderUserDefinedIncludes：R
E_SetLightState）ENABLED START */
/* Start of user defined includes  -Do not remove this
comment */
/* End of user defined includes-Do not remove this com
ment */
/*PROTECTED REGION END */

#define RE_SetLightState_START_SEC_CODE
#include "IOAbstractionSWC_MemMap.h"
FUNC（void , IOAbstractionSWC_CODE）RE_SetLightState//r
eturn value & FctID
（
     VAR（UInt8，AUTOMATIC）DESetLightState
）
{

   /* Local Data Declaration */

   /*PROTECTED REGION ID（UserVariables：RE_SetLightSta
te）ENABLED START */
   /* Start of user variable defintions-Do not remove
```

```
this comment  */
    /* End of user variable defintions-Do not remove th
is comment  */
        /*PROTECTED REGION END */
    Std_ReturnType retValue = RTE_E_OK;
    /*  ----------------------------------------Data Read
----------------------------------------  */
    /*  ----------------------------------------Server Ca
ll Point  ---------------------------------  */
    /*  ----------------------------------------CDATA----
----------------------------------------  */
    /*  ----------------------------------------Data Writ
e----------------------------------------  */
    /*  ----------------------------------------Trigger I
nterface----------------------------------  */
    /*  ----------------------------------------Mode Mana
gement------------------------------------  */
    /*  ----------------------------------------Port Hand
ling--------------------------------------  */
    /*  ----------------------------------------Exclusive
 Area--------------------------------------  */
    /*  ----------------------------------------Multiple
Instantiation----------------------------  */
    /*PROTECTED REGION ID（User Logic：RE_SetLightState
）ENABLED START */
    /* Start of user code-Do not remove this comment */
    /* End of user code-Do not remove this comment */
    /*PROTECTED REGION END */
}

#define RE_SetLightState_STOP_SEC_CODE
#include "IOAbstractionSWC_MemMap.h"
```

5.4　基于ISOLAR-A的系统级设计与配置方法

如前所述，AUTOSAR支持整车级别的软件架构设计，开发人员可以进行整车级别的软件组件定义，再将这些软件组件分配到各个ECU中，这就是AUTOSAR系统级设计需要完成的主要任务。下面结合AUTOSAR方法论的相关概念，对基于ISOLAR-A工具的AUTOSAR系统级设计方法进行讲解。

5.4.1　系统配置输入文件创建与导入

从AUTOSAR方法论中可以看到，系统级设计是基于系统配置输入描述文件来进行的，系统配置输入描述文件主要包括：

①软件组件描述文件；

②ECU资源描述文件；

③系统约束描述文件。

其中，软件组件描述文件主要包含每个软件组件的设计信息，如端口接口、端口、运行实体、RTE事件等，它们可以通过Matlab/Simulink基于软件组件模型进行自动生成，亦可以在ISOLAR-A工具中直接进行软件组件相关元素的设计，这些方法在前文中都已经基于具体案例进行了介绍。

ECU资源描述文件描述了ECU的硬件资源需求，如处理器、存储器、传感器等，这些描述主要体现在硬件原理图等文件中。

系统约束描述文件主要描述了总线信号等信息，其中与总线相关的信息在ISOLAR-A中可以基于传统网络描述文件进行导入。本书所涉及的示例是基于CAN总线的系统，所以这里以DBC文件为例进行讲解。在

系统级设计工具ISOLAR-A中可以导入DBC文件，并可以将其中的信息转化为arxml格式。每个DBC文件描述了一个CAN网络内各节点的通信信息，所以对于同一个ECU，若它涉及多个CAN网络，则每个CAN网络都需要一个DBC文件来对其进行描述。下面扼要介绍DBC文件的建立方法，这里使用Vector公司的CANdb++工具来进行DBC文件建立。整个DBC文件基本需要配置如下信息（图5.65）：

图5.65　DBC文件信息列表

①Networks（网络）；

②ECUs（电控单元）；

③Network nodes（网络节点）；

④Messages（报文）；

⑤Signals（信号）。

如前所述，每个DBC描述的是一个网络上的通信信息，所以首先要定义一个Networks（网络），这在新建database时自动创建出来。之后

就是定义Network nodes（网络节点），右键点击Network nodes新建即可，同时将在ECUs（电控单元）菜单中自动创建出相关ECU信息。

定义了网络节点后，就需要根据项目需求完成Signal（信号）与Message（报文）的设计。首先，进行信号添加，右键点击Signals新建，将弹出如图5.66所示的界面，其中主要可以定义信号的如下属性。

图5.66 Signal定义

①Length（长度）。

②Byte Order（字节顺序）：Intel（英特尔模式）以及Motorola（摩托罗拉模式）。

③Value Type（数据类型）：Signed、Unsigned、IEEE Float、IEEE Double。

④Scaling（缩放比例）：Factor（比例系数）、Offset（偏移量）。

⑤Minimum（最小值）、Maximum（最大值）。

⑥Value Table（数值表）。

在定义了所有信号之后，即可进行报文的设计。右键点击Message新建即可，主要需要定义如下属性（图5.67）。

Message 'Light_State (0x2)'

Definition | Signals | Transmitters | Receivers | Layout | Attributes | Comment

Name: Light_State

Type: CAN Standard

ID: 0x2

DLC: 2

Transmitter: LightECU

Tx Method: Cyclic

Cycle Time: 100

确定 | 取消 | 应用(A) | 帮助

Message 'Light_State (0x2)'

Definition | Signals | Transmitters | Receivers | Layout | Attributes | Comment

Multiplexor Signal: -- No Multiplexor --

	7	6	5	4	3	2	1	0
0	LightType msb 7	6	5	4	3	2	1	lsb 0
1	LightState msb 15	14	13	12	11	10	9	lsb 8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	63	62	61	60	59	58	57	56

Arrange

To Front

To Back

Add...

Remove

Bit index

☐ Inverted

确定 | 取消 | 应用(A) | 帮助

图5.67　Message定义

①Type（报文类型）：CAN Standard（标准帧）、CAN Extended（扩展帧）。

②Id：报文的Id号。

③DLC（Data Length Code）：报文长度。

再切换到Layout界面就可以完成报文中信号位置的设置。

在完成所有的信号以及报文设计之后，就需要将它们在各节点中的交互关系定义出来，即将各报文添加到相应节点。若报文是发送的，则将该报文添加到相应CAN节点的Tx　Messages即可；若报文是接收的，则需要拖动该报文的信号，将它们逐一添加到Mapped　Rx　Signals列表中。

最终，按照示例开发需求，可以完成系统的DBC文件的建立，其中包含三个网络节点，即DisplayECU、LightECU与SensorECU，包含两帧报文，即Light_Intensity与Light_State，如图5.68所示。

图5.68　示例DBC文件建立结果

之后，就可以将系统配置所需要的输入文件导入ISOLAR-A系统级设计工具。其中，对于软件组件描述文件，只要将其放在AUTOSAR工

程文件夹下即可，打开ISOLAR-A工具将自动识别其中的信息。而对于传统CAN网络描述文件DBC文件，则需要在ISOLAR-A工具中进行导入，其方法描述如下。

首先，点击ISOLAR-A主菜单栏中的“D”DBC Importer，如图5.69所示。

图5.69　DBC文件导入（一）

将弹出如图5.70所示的界面，点击Browse，选择先前创建的DBC文件。此时，会自动完成系统描述文件的选择与AR Package的选择。由于DBC中包含三个ECU，这里分别为它们新建一个CAN Controller，点击Next可进入如图5.71所示的界面。

图5.70　DBC文件导入（二）

图5.71　DBC文件导入（三）

此时，需要选择DBC文件中所定义的网络节点，即ECU。为之后体现系统级设计的特点，这里勾选三个ECU，点击Next，将弹出如图5.72所示的界面。

图5.72　DBC文件导入（四）

在选择了ECU的基础上，需要选择与其相关的CAN报文，点击Select All，再点击Finish即可完成——DBC文件的导入。

DBC文件导入成功之后，在ISOLAR-A工具中就可以看到通信相关的PDU与Signal的信息，如图5.73所示，这些是之后配置CAN通信协议栈的基础信息。另外，还有ECU Instance与CAN Network信息，如图5.74与图5.75所示。

图5.73　Signal与PDU信息

图5.74　ECU Instance与CAN Network信息

图5.75　System Editor界面

5.4.2 Composition SWC建立

按照AUTOSAR方法论的要求，需要进行系统级设计，这可以是一个整车级的软件架构设计。本书以车灯控制系统为例基于ISOLAR-A工具来展现AUTOSAR系统级设计“自顶向下”的魅力。这里先建立一个VehicleComposition部件来包含整车级别的所有软件组件，在本书示例中则包含车灯控制系统所涉及的软件组件。

如前所述，AUTOSAR软件组件大体上可分为原子软件组件（Atomic SWC）和部件（Composition SWC）。部件虽然也属于软件组件的范畴，但其建立方法和思想与原子软件组件有所不同。

右键点击Software→Create Compositions→Elements|Composition Sw Component Typ如图5.76所示，会弹出如图5.77所示界面，输入相关信息，点击Finish即可。

图5.76　Composition SWC建立（一）

图5.77　Composition SWC建立（二）

双击新建的VehicleComposition可进入Composition设计界面，点击“+”ADD可以添加软件组件，如图5.78所示。本书中将所有SWC都加入到VehicleComposition，点击OK即可，结果如图5.79所示。

图5.78　Composition中添加SWC

图5.79　Composition中添加SWC结果

在添加完SWC之后，就需要连接各通信端口，即添加Assembly Connectors。右键点击VehicleComposition→Open With→Auto Assembly Connect Wizard，如图5.80所示。

图5.80　Composition中添加Assembly Connectors（一）

在弹出的界面中，可以勾选需要连接的端口，确认后点击Finish即可，如图5.81所示。

图5.81　Composition中添加Assembly Connectors（二）

至此，整车级别的Composition SWC建立完成。下面展现VehicleComposition的建立结果，图5.82（上）为Composition Overview界面；图5.82（中）为AR Explorer Software→Compositions界面；图5.82（下）为Prototype Editor界面，点击图5.82（中）VehicleComposition→Open With→Prototype Editor即可打开。

*VehicleComposition
Composition Overview
Matrix View
Composition Hierarchy
ADD
DELETE
type filter text
VehicleComposition
CPT_EcuAliveIndicatorSWC
CPT_IOAbstractionSWC
CPT_LightControlSWC
CPT_SensorSWC
CPT_LightRequestSWC
CPT_DisplaySWC
CPT_EcuBaseSWC
Connectors from VehicleComposition
Showing: 9/9
Filter :
RE
Valid... Type Short Name Source CPT Source Port Target CPT Target Port
1 ASC_CPT_IOA... CPT_IOAbst... PPortEcuAli... CPT_EcuAliv... RP_EcuA
2 ASC_CPT_IOA... CPT_IOAbst... PPortGetBu... CPT_LightR... RPortGe
3 ASC_CPT_IOA... CPT_IOAbst... PPortGetLig... CPT_LightC... RPortGe
4 ASC_CPT_IOA... CPT_IOAbst... PPortSetLig... CPT_LightC... RPortSet
5 ASC_CPT_Ligh... CPT_LightC... PPortSendL... CPT_Displa... RPortSe
6 ASC_CPT_Ligh... CPT_LightC... PPortSendL... CPT_Displa... RPortSe
7 ASC_CPT_Ligh... CPT_LightR... PPortContr... CPT_LightC... RPortCo
8 ASC_CPT_Ligh... CPT_LightR... PPortNoCo... CPT_EcuBas... RPortNc
9 ASC_CPT_Sen... CPT_Sensor... PPortGetLig... CPT_LightR... RPortGe
Compositions
VehicleComposition
Component Prototypes
CPT_DisplaySWC
CPT_EcuAliveIndicatorSWC
CPT_EcuBaseSWC
CPT_IOAbstractionSWC
CPT_LightControlSWC
CPT_LightRequestSWC
CPT_SensorSWC
Assembly Connectors
ASC_CPT_IOAbstractionSWC_PPortEcuAliveIndicator_CPT_EcuAliveIndicatorSWC_RP_EcuAliveIndicator
ASC_CPT_IOAbstractionSWC_PPortGetButtonState_CPT_LightRequestSWC_RPortGetButtonState
ASC_CPT_IOAbstractionSWC_PPortGetLightState_CPT_LightControlSWC_RPortGetLightState
ASC_CPT_IOAbstractionSWC_PPortSetLightState_CPT_LightControlSWC_RPortSetLightState
ASC_CPT_LightControlSWC_PPortSendLightState_CPT_DisplaySWC_RPortSendLightState
ASC_CPT_LightControlSWC_PPortSendLightType_CPT_DisplaySWC_RPortSendLightType
ASC_CPT_LightRequestSWC_PPortControlLightCmd_CPT_LightControlSWC_RPortControlLightCmd
ASC_CPT_LightRequestSWC_PPortNoComRequest_CPT_EcuBaseSWC_RPortNoComRequest
ASC_CPT_SensorSWC_PPortGetLightIntensity_CPT_LightRequestSWC_RPortGetLightIntensity

图5.82

图5.82　VehicleComposition建立结果

5.4.3　系统配置

在完成了VehicleComposition建立后，可以进行系统配置。右键点击System→Create System Info→Elements|System（图5.83）可以新建System Info，命名为VehicleSystem，结果如图5.84所示。

图5.83　System Info新建（一）

图5.84　System Info新建（二）

新建System Info VehicleSystem后，需要为其添加一个Compositon，右键点击VehicleSystem→New Child→Root Software Compositions|Root Sw Compos 5.85），并在SoftwareComposition中引用先前建立的VehicleComposition即可，如图5.86所示。

图5.85　VehicleSystem中Compositon添加（一）

图5.86　VehicleSystem中Compositon添加（二）

在建立一个整车级的系统后，就需要完成软件组件到ECU的映射（SWC To ECU Mapping），右键点击VehicleSystem→Open With→SWC To ECU Mapping Editor，如图5.87所示。

图5.87　SWC To ECU Mapping

打开SWC To ECU Mapping Editor界面后，点击Create System Mapping可以添加系统映射。此时，可将VehicleComposition中的软件组件依次映射到相应的ECU，这就是AUTOSAR方法论中所提出的从整车级的软件组件定义过渡到具体ECU开发的关键环节。本书示例中，按照前期系统定义，可将各SWC分别拖动到DisplayECU、LightECU与SensorECU完成映射，如图5.88所示。

图5.88　SWC To ECU Mapping结果

完成SWC To ECU Mapping后，在系统配置阶段还需要完成系统信号与端口数据元素的映射，即System Data Mapping。右键点击VehicleSystem→Open With→System Data Mapping Editor，如图5.89所示。

图5.89　System Data Mapping（一）

打开System Data Mapping Editor后，根据需求选择相应端口，完成系统信号与端口数据元素的映射即可，如图5.90所示。

System Data Mapping

Root Software Composition configured in System: VehicleComposition　Compute Nets for the System: Compute Nets

Filters Information

Signal Data mapping

This section contains Signal Data Mapping Information.

Showing: 21/21 | Filter :

	Component Info	Data Element Info	Signal Info	Pdu info	Frame Info
	Component Prototype	Port Prototype	System Signal/Signal Group	ISignal to IPdu Mapping	Pdu to Frame Mapping
4	CPT_IOAbstractionSWC		NA	NA	NA
5	CPT_LightControlSWC		NA	NA	NA
6		PPortSendLightState	SS:LightState	LightState	Light_State
7		PPortSendLightState			
8		PPortSendLightType			
9		PPortSendLightType	SS:LightType	LightType	Light_State
10		RPortControlLightCmd			
11	CPT_LightRequestSWC		NA	NA	NA
12		PPortControlLightCmd			
13		PPortNoComRequest			
14		RPortGetLightIntensity	SS:LightIntensity	LightIntensity	Light_Intensity
15	CPT_SensorSWC		NA	NA	NA

图5.90　System Data Mapping（二）

5.4.4　ECU信息抽取

本书示例需要开发的目标ECU是LightECU，所以在完成了系统配置之后，就需要进行待配置ECU信息抽取（ECU Extract）。右键点击Ecus文件夹中的LightECU→Create　ECUExtract（图5.91），将弹出如图5.92所示的界面，点击Finish即可。

图5.91　ECU Extract（一）

图5.92　ECU Extract（二）

由于本书示例是针对LightECU进行开发的，所以将ECU　Extract生成在同一工程文件夹下。最终，ECU信息抽取结果如图5.93所示。该过程中将自动创建名为LightECU_FlatView 的Composition，该Composition中包含了分配到LightECU的所有SWC。

图5.93　ECU Extract结果

此时，可以查看LightECU_FlatView　Composition，ISOLAR-A工具在ECU　Extract过程中已经将与LightECU无关的SWC去除了，原来ECU内（Intra-ECU）通信的端口保留了Assembly Connectors的属性；而原来ECU间（Inter-ECU）通信的端口已经成为Delegation　Connectors，如图5.94所示。

图5.94　LightECU_FlatView

5.5 本章小结

本章主要介绍了基于ETAS ISOLAR-A工具的AUTOSAR系统级设计与配置方法。首先，介绍了ISOLAR-A工具的安装方法及操作界面。之后，结合AUTOSAR软件组件的相关概念，详细讲解了基于ISOLAR-A工具的软件组件设计方法。进而对基于ISOLAR-A工具的AUTOSAR系统级设计与配置方法进行了详细的介绍。通过本章的学习，可以巩固AUTOSAR软件组件与AUTOSAR方法论中系统级开发的相关理论知识，并学会在ISOLAR-A工具中的基本实现方法。

第6章　AUTOSAR ECU级开发之RTE与BSW（除MCAL外）

先前已经完成了本书示例系统级与软件组件级的设计，根据AUTOSAR方法论，还需要进行目标ECU的配置，即ECU级设计，该阶段主要是对运行时环境（RTE）和基础软件层（BSW）模块的配置。对于基础软件层，如先前AUTOSAR分层架构图所示，其中包含很多模块，根据实际需求可以进行选择性配置。

结合本书示例需求，A型和B型车灯控制器所使用的BSW模块如下：系统服务层中的操作系统（Operating System，OS）、基础软件模式管理器（Basic Software Mode Manager，BswM）、ECU状态管理器（ECU State Manager，EcuM）、通信管理模块（Communication Manager，ComM）；通信服务中的通信模块（Communication，Com）、CAN状态管理模块（CAN State Manager，CanSM）、协议数据单元路由模块（PDU Router，PduR）；ECU抽象层中的I/O硬件抽象，通信硬件抽象中的CAN接口模块（CAN Interface，CanIf）；微控制器抽象层中的MCU驱动、GPT驱动，通信驱动中的CAN驱动，I/O驱动中PORT驱动、DIO驱动、ADC驱动、PWM驱动、ICU驱动。其中，I/O硬件抽象层作为AUTOSAR中的非标准模块，其实现方法已在前面章节介绍，本章不再重复。

下面先介绍ETAS针对AUTOSAR ECU级开发推出的工具，并基于这些工具介绍AUTOSAR ECU级开发中除微控制器抽象层MCAL以外的模块概念与配置方法。

6.1 ETAS RTA系列工具简介

ETAS RTA系列工具是针对AUTOSAR ECU级的设计与开发工具，主要包括RTA-BSW（AUTOSAR基础软件）、RTA-RTE（AUTOSAR运行时环境生成器）和RTA-OS（AUTOSAR实时操作系统）。

6.1.1 RTA-BSW简介

RTA-BSW是一套高质量的基础软件，可为先进的汽车电子控制单元（ECU）提供全面的AUTOSAR R4.x中间件。RTA-BSW可提供一套全面的AUTOSAR栈，包括通信、存储、诊断、标定和复杂驱动等。基于长时间在软件平台开发方面的经验，RTA-BSW可为ECU应用开发提供全面的AUTOSAR R4.x平台。它们易于配置、集成和测试，支持将应用运行于真实的ECU硬件或者虚拟ECU平台（如ISOLAR-EVE）。RTA-BSW适用于符合AUTOSAR规范的ECU软件，可用于汽车动力系统、底盘控制系统、电池管理系统、高级驾驶辅助系统、车身电子控制系统等。

RTA-BSW的特点及优势如下：

①RTA-BSW是基于ISO 26262功能安全标准中ASIL-D的流程开发而成的；

②由于全面支持AUTOSAR R4.x，RTA-BSW可为ECU软件开发项目提供即用型解决方案；

③除了AUTOSAR标准模块外，RTA-BSW还包括一些对标准AUTOSAR模块的功能扩展；

④RTA-BSW支持自动配置和代码生成，可最大限度地减少开发符合AUTOSAR规范的基础软件的工作量；

⑤借助RTA-BSW的自动测试功能，可在诸如ISOLAR-EVE等虚拟ECU平台上进行软件前期验证；

⑥根据与主要芯片微控制器抽象层MCAL供应商签订的协议，MCAL可与RTA-BSW一起使用，从而构建一套完整的BSW。

6.1.2 RTA-RTE简介

RTA-RTE（AUTOSAR运行时环境生成器）可为符合AUTOSAR规范的ECU软件提供运行时环境。它可以生成符合AUTOSAR R4.x和R3.x的运行时环境；提供配置生成运行时环境的多种选择；可以检测arxml文件的正确性，以确保开发过程的高质量；可输出操作系统配置文件，以集成运行时环境和操作系统。

使用RTA-RTE的优势如下：

①RTA-RTE是一套已通过ISO 26262（ASIL-D）认证的AUTOSAR运行时环境生成器；

②用RTA-RTE生成的运行时环境是独立于目标ECU的MISRA C代码，可以在不同开发环境和ECU平台使用；

③可以优化运行时环境，精确匹配应用需求；

④与各种类型的编译器和目标ECU硬件兼容，可以在广泛的ECU平台上使用；

⑤通过使用虚拟功能总线（VFB）进行跟踪，易于查错。

6.1.3 RTA-OS简介

RTA-OS是用于嵌入式系统的实时操作系统（Real Time Operating System，RTOS）。它符合AUTOSAR、OSEK/VDX、ISO 26262和MISRA C最新版标准。

RTA-OS以ETAS推出多年的RTA-OSEK实时操作系统开发经验为基础，它可用于单核与多核单片机，并且资源开销小。由于其通过了ISO 26262 ASIL-D的认证，可以被用于安全要求苛刻的控制器。

RTA-OS可以与RTA-RTE和RTA-BSW结合使用，以开发出一个符合AUTOSAR规范的软件平台，但RTA-OS也可以用于非AUTOSAR软件。此外，基于ISOLAR-EVE虚拟ECU平台，即使在目标ECU硬件还不可用的情况下也可以对OS进行虚拟验证。

6.2 ETAS RTA系列工具入门

6.2.1 RTA系列工具安装方法

本书中需要用到ETAS RTA-RTE、RTA-BSW和RTA-OS。由于RTA-RTE与RTA-OS的安装方法和安装过程中的注意点类似，所以这里着重介绍RTA-RTE与RTA-BSW的安装方法。

（1）ETAS RTA-RTE的安装方法

首先，打开RTA-RTE安装包里的autostart.exe，启动安装，会弹出如图6.1所示的界面。点击Installation后，会显示如图6.2所示的界面。

图6.1　ETAS RTA-RTE安装步骤（一）

图6.2　ETAS RTA-RTE安装步骤（二）

点击RTA-RTE　Installation，会弹出如图6.3所示的界面，进入安装向导，点击下一步（Next），进入如图6.4所示的界面。

图6.3　ETAS RTA-RTE安装步骤（三）

图6.4　ETAS RTA-RTE安装步骤（四）

此时，用户可以选择RTA-RTE的安装路径，在文件夹命名时可以附加一些软件的版本信息，例如，这里使用的是RTA-RTE v6.0.0，则安装文件夹可命名为RTA-RTE_V600，这便于今后在多版本共存情况下进行管理。点击下一步Next将弹出如图6.5所示的界面。

图6.5　ETAS RTA-RTE安装步骤（五）

这里，用户可以根据需求安装相应工具，初次安装建议选择Custom Installation，下面也可以通过勾选需要安装的子工具。确认了需要安装的工具后，可以点击下一步（Next），会弹出如图6.6所示的界面。

图6.6　ETAS RTA-RTE安装步骤（六）

在确认了软件快捷方式生成路径后，点击下一步（Next），会弹出如图6.7所示的界面。

图6.7　ETAS RTA-RTE安装步骤（七）

再次确认安装路径以及需要安装的工具后，点击Install进入如图6.8所示的界面。

图6.8　ETAS RTA-RTE安装步骤（八）

当显示如6.9所示的界面，表明安装完成，点击Finish即可。至此，RTA-RTE安装完成。

图6.9　ETAS RTA-RTE安装步骤（九）

（2）ETAS RTA-BSW的安装方法

如前所述，ETAS RTA-BSW是ISOLAR-A工具的插件，所以它的安装和传统Eclipse插件安装方法类似。为便于用户使用，ETAS对整个安装过程做了一键式封装，用户只需要打开安装文件中的脚本install.bat，按照提示操作即可，如图6.10所示。这里需要指出，RTA-BSW安装之前需要保证计算机上已安装ETAS ISOLAR-A工具。

图6.10　ETAS RTA-BSW安装步骤

6.2.2　RTA系列工具界面说明

如前所述，ISOLAR-A中可添加BCT（ECU Conf+Code Generation）附件，如图6.11所示。该界面大致上分为三大块：左边ECU Conf Navigator展示了已配置的BSW模块；中间Outline可以进行BSW模块的添加与新配置项的添加；右边界面则通过表格等形式列出了各模块的配置参数子项。

图6.11　集成RTA-BSW之后的ISOLAR-A BCT界面

RTA-OS工具界面如图6.12所示，其界面布局与ISOLAR-A类似。

图6.12　RTA-OS工具界面

6.3　CAN通信协议栈概念与配置方法介绍

6.3.1　CAN通信协议栈概念

AUTOSAR通信栈位于运行时环境（RTE）与微控制器抽象层（MCAL）之间，其可以简化ECU间的通信服务，实现不同类型或速率总线间的数据交互，如图6.13所示。在AUTOSAR通信栈中，位于服务层的有通信模块（Communication，Com）、诊断通信管理模块（Diagnostic Communication Manager，Dcm）、协议数据单元路由模块（Protocol Data Unit Router，PduR）、协议数据单元复用模块（I-PDU Multiplexer，IpduM）、总线相关的传输模块（如CanTp、FrTp等）以及通信和网络管理相关的模块；位于ECU抽象层的是与总线相关的接口模块（如CanIf、LinIf等）；位于微控制器抽象层的是与总线相关的驱动模块（如Can、Lin等）。

图6.13　AUTOSAR通信栈结构

AUTOSAR通信栈对应用层隐藏了与总线相关的协议和报文的属性。以基本的CAN通信为例，其发送机制如图6.13中1→2→3所示，该过程描述如下：

①Com模块获取应用层的信号（Signal），经一定处理封装为I-PDU（Interaction Layer Protocol Data Unit）发送到PduR模块；

②PduR根据路由协议中所指定的I-PDU目标接收模块，将接收到的I-PDU经一定处理后发送给CanIf；

③CanIf将信号以L-PDU（Data Link Layer Protocol Data Unit）的形式发送给CAN驱动模块。

最终，实现了基于CAN总线的基本数据发送；反之亦然。

6.3.2 CAN通信协议栈配置方法

在先前AUTOSAR系统级设计与配置阶段已经介绍了，对基于CAN总线的系统可以在ISOLAR-A中通过导入DBC文件来进行通信相关信息的获取。安装了RTA-BSW工具并集成于ISOLAR-A工具之后，在ISOLAR-A主界面中，可以点击如图6.14所示的RTA-BSW Configuration Generation按钮，进行CAN通信协议栈中Com、PduR、CanIf、ComM、CanSM等模块的预配置。

图6.14 RTA-BSW Configuration Generation

点击RTA-BSW Configuration Generation按钮后，会弹出如图6.15所示的窗口，选择待开发ECU的ECU Instance，点击OK即可。

图6.15　ECU Instance选择

在完成预配置之后，切换到BCT（ECU Conf+Code Generation）界面就可以看到一些模块的配置信息，如图6.16所示。基于这些预配置的模块就可以开始ECU级开发。

图6.16　RTA-BSW Configuration Generation结果

虽然CAN通信协议栈相关模块的配置基本都可以自动生成，但为便于读者理解，这里还是着重介绍其中相关模块的概念与一些重要配置项的意义。主要涉及EcuC、Com、PduR、CanIf、ComM、CanSM，由于Can模块，即CAN驱动模块属于微控制器抽象层MCAL的范畴，所以将在后面MCAL配置章节单独讲解。

（1）EcuC模块

数据在CAN通信协议栈各层间都是以PDU（Protocol Data Unit）形式传输的，为了将各层PDU关联起来，则需要定义全局PDU（Global PDU）。由于全局PDU不属于任何一个标准BSW模块，所以AUTOSAR提出了一个EcuC模块来收集一些配置信息。

在ECU Conf Navigator界面，右键点击EcuC“EcuC”→Open In Editor，如图6.17所示。在Outline界面可以看到EcuC模块的具体配置，如图6.18所示。

图6.17　EcuC模块配置打开方法

图6.18　EcuC模块配置

可见，在EcuC模块中定义全局PDU时不需要关心其数据类型，只需要定义PDU长度即可。见图6.19。

图6.19　EcuC中全局PDU配置

（2）Com模块

Com模块位于运行时环境RTE与PduR模块之间，其主要功能包括：

①将信号装载到I-PDU中发送，从接收到的I-PDU中解析出信号；

②提供信号路由功能，将接收到的I-PDU中的信号打包到发送I-PDU中；

③通信发送控制（启动/停止I-PDU组）；

④发送请求的应答等。

先前生成的Com模块配置如图6.20所示，其中主要有ComIPdus（I-PDU）、ComIPduGroups（I-PDU工作组）、ComSignals（信号）、ComTimeBases（时间基准）等。

图6.20　Com模块配置

ComIPduGroups配置如图6.21所示，其中创建了两个I-PDU工作组，即ComIPduGroup_Tx和ComIPduGroup_Rx，其Id分别为0和1。

ComIPduGroups

		Short Name	ComIPduGroupHandleId
Filter			
0	☐	ComIPduGroup_Rx	0
	☐	ComIPduGroup_Tx	1

图6.21　ComIPduGroups配置

其次，Com层中的Signal是应用层通过Com模块收发的基本单元，也是Com层内信息交互的基本单元，其需要引用系统信号（System　Signal）。I-PDU作为Com层与下层网络交互的基本单元，可由一个或多个Signal信号组成，各信号在Com模块中装载和解析。

每个ComSignal需要配置信号的初始值（ComSignalInitValue）、发送属性（ComTransferProperty）、数据类型（ComSignalType）、字节顺序（ComSignalEndianness）、字节大小（ComSignalLength）、系统信号引用（ComSystemTemplateSystemSignalRef）等。对于发送属性主要可分为PENDING、TRIGGERED和TRIGGERED_ON_CHANGE，它们的特点描述如下。

①PENDING：写入信号值不能触发该信号相关的I-PDU的发送。

②TRIGGERED：根据信号相关的I-PDU的发送模式，写入该信号值可以触发该信号相关的I-PDU的发送。

③TRIGGERED_ON_CHANGE：根据信号相关的I-PDU的发送模式，当写入该信号值，并且该信号值有变化时，才会触发该信号相关的I-PDU的发送。

此外，为了保证应用层发送的复杂类型数据（如结构体）的完整性，有时还可以定义信号组（Com Signal Group），免去数据过于复杂需要拆解而导致的完整性破坏。本书示例ComSignals配置如图6.22所示。

图6.22　ComSignals配置

对于每个Com I-PDU需要设定I-PDU的传输方向（ComIPduDirection）、信号处理方式（ComIPduSignalProcessing）、类型（ComIPduType）、所属的I-PDU工作组（ComIPduGroupRef）、Com信号引用（ComIPduSignalRef）、全局PDU引用（ComPduIdRef）等。最终配置如图6.23所示。

图6.23　ComIPdus配置

（3）PduR模块

PduR模块是主要为通信接口模块、传输协议模块、诊断通信管理模块以及通信模块提供基于I-PDU的路由服务。它在通信协议栈中起着承上启下的作用，为上层服务基础软件模块和应用屏蔽了网络细节，使得上层基础软件模块和应用不用关心运行于哪种总线网络之上。同时，PduR模块提供了基于I-PDU的网关功能，使得不同总线之间的通信成为可能。PduR模块自动生成的配置如图6.24所示。

图6.24　PduR模块配置

在PduR模块中首先需要添加PduRBswModules，即添加所用到的通信协议栈中的相关模块，并勾选相关属性。本书示例使用到了Com模块和CanIf模块，PduRBswModules配置如图6.25所示。

图6.25　PduRBswModules配置

另外，需要定义路由路径（PduRRoutingPaths）。路由路径为源PDU（Source PDU）到目标PDU（Destination PDU）的描述，它们都需要通过引用前述EcuC中定义的全局PDU来进行关联，如图6.26所示。

图6.26　全局PDU引用

最终，本书示例PduR模块的PduRRoutingPaths配置如图6.27所示。

图6.27　PduRRoutingPaths配置

（4）CanIf模块

CAN接口层（CanIf）是访问CAN总线的标准接口。CanIf抽象了CAN控制器的位置信息，并向上提供了一个与平台无关的接口，即上层不用关心CAN控制器是微控制器的片上设备还是片外设备。

CanIf模块主要功能：完成对CanIf和控制器中全局变量及配置缓冲

区的初始化；发送请求服务，提供供上层应用在CAN网络上发送PDU的接口；发送确认服务，发送成功后通知上层，或者发送取消确认后存于发送缓存；接收指示服务，成功接收PDU后通知上层。

CanIf模块主要配置为硬件对象句柄（Hardware object handle，Hoh），包括Hth（Hardware transmit handle）和Hrh（Hardware receive handle），它们需要引用Can模块中定义的CAN硬件对象（CanHardwareObject），CanHardwareObject是对CAN邮箱（MailBox，MB）的抽象，在后续MCAL配置中会进行讲解。CanIf模块还需要配置CanIf层的PDU，每个PDU需要引用一个Hth或者Hrh，即完成PDU向MB的分配。CanIf模块主要配置如图6.28～图6.30所示。

图6.28　CanIf模块配置

图6.29　Hrh和Hth配置

图6.30　CanIfRxPduCfgs配置

（5）ComM模块

通信管理模块（Communication Manager，ComM）可以简化总线通

信栈的初始化、网络管理等，并可收集/协调总线通信访问请求。它主要提供了三种通信模式。

①COMM_FULL_COMMUNICATION：FULL通信模式，此状态既能接收又能发送。

②COMM_SILENT_COMMUNICATION：SILENT通信模式，此状态只能进行接收。

③COMM_NO_COMMUNICATION：NO通信模式，此状态不能进行通信。

本书示例中ComM模块的配置生成结果如图6.31所示。

图6.31　ComM模块配置

ComMChannel配置中可以配置ComMMainFunction（）周期，默认为10ms，如图6.32所示。

图6.32　ComMChannel配置

（6）CanSM模块

CAN状态管理器（CAN State Manager，CanSM）负责实现CAN网络控制流程的抽象，它为ComM模块提供API来请求CAN网络进行通信模式的切换，其配置生成如图6.33所示。

图6.33　CanSM模块配置

CanSMGeneral配置中可以配置CanSMMainFunction（）周期，默认为10ms，如图6.34所示。

图6.34　CanSMGeneral配置

6.4 EcuM模块概念与配置方法介绍

EcuM（ECU State Manager）模块属于系统服务层，它在系统服务层中的具体位置如图6.35所示。

图6.35 EcuM模块具体位置

EcuM模块负责初始化（Initialize）和反初始化（De-initialize）一些BSW模块。AUTOSAR ECU模式管理分为Fixed和Flexible两种方式，Fixed有如下确定的模式：

①STARTUP；

②RUN；

③POST_RUN；

④SLEEP；

⑤WAKE_SLEEP；

⑥SHUTDOWN。

Flexible模式则允许其他的情况，如快速/分部（Fast/Partial）启动、多核管理（Multicore Management）等。

另外，EcuM模块还可以配置ECU睡眠模式（Sleep Modes）、下电原因（Shutdown Causes）、复位模式（Reset Modes），管理所有ECU唤醒源（Wakeup Sources）。

在BCT界面中可以完成EcuM模块的添加和配置。在Outline界面中右键点击
Ecu State Manager Module→Create'EcuM'with mandatory containers with va可新建EcuM模块的配置，如图6.36所示。最终，本书示例的EcuM模块配置总览如图6.37所示。

图6.36　EcuM模块配置新建

图6.37　EcuM模块配置

如前所述，EcuM模块需要初始化一些BSW模块，所以需要定义一系列初始化列表。对于Fixed模式，可以定义四个初始化列表：

①初始化列表0（Driver Init List Zero）；

②初始化列表1（Driver Init List One）；

③初始化列表2（Driver Init List Two）；

④初始化列表3（Driver Init List Three）。

其中，列表0和1在操作系统OS启动之前完成，而列表2和3则需要操作系统OS支持，故在其启动之后完成。对于Flexible模式，EcuM只需要完成列表0和1中各模块的初始化，而后两个列表中的模块初始化需要由BswM模块来实现。这里给出一个推荐的模块初始化顺序列表，如表6.1所示，也可根据实际情况做出相应调整。

表6.1　BSW模块初始化顺序列表

Driver Init List Zero	Driver Init List One	Driver Init List Two	Driver Init List Three
Det	Dem Pre-Initialization	Spi	ComM
	Mcu	EEP	Dem
	Port	Fls	Fim
	Dio	NvM (start of ReadAll)	RTE②
	Gpt	CAN Stack	Applications②
	Watchdog Stack	LIN Stack	
	Adc	FR Stack	
	Icu	PduR	
	Pwm	Nm	
	Ocu	IpduM	
	BswM①	Com	
		Dcm	

①用于Fixed EcuM。
②用于Flexible EcuM。

根据EcuM模块的作用，可以完成该模块的配置，这里使用Flexible EcuM。

（1）EcuMDriverInitListZero配置

本书示例EcuMDriverInitListZero配置如图6.38所示。

EcuMDriverInitItems

		Short Name	EcuMModuleID	EcuMModuleService
Filter				
0	☐	Det_Init	Det	Init

图6.38　EcuMDriverInitListZero配置

（2）EcuMDriverInitListOne配置

本书示例EcuMDriverInitListOne配置如图6.39所示。

EcuMDriverInitItems

		Short Name	EcuMModuleID	EcuMModuleService	EcuM...osar	Ecu...ble	EcuMRbModuleConfigDataSetName
Filter							
0	☐	Mcu_Init	Mcu	Init	True	True	Mcu_ConfigRoot
	☐	McuFunc_InitializeClock	McuFunc	InitializeClock	True	False	
2	☐	Port_Init	Port	Init	True	True	Port_ConfigRoot
3	☐	Dio_Init	Dio	Init	True	True	Dio_ConfigRoot
4	☐	Gpt_Init	Gpt	Init	True	True	Gpt_ConfigRoot
5	☐	Adc_Init	Adc	Init	True	True	Adc_ConfigRoot
6	☐	Can_Init	Can	Init	True	True	Can_ConfigRoot
7	☐	Icu_Init	Icu	Init	True	True	Icu_ConfigRoot
8	☐	Pwm_Init	Pwm	Init	True	True	Pwm_ConfigRoot

图6.39　EcuMDriverInitListOne配置

为了便于理解初始化列表的实现过程，这里先展现一下最终生成的初始化列表EcuMDriverInitListZero和EcuMDriverInitListOne的部分代码。其中，EcuM_AL_DriverInitOne函数中所调用的各MCAL模块初始化函数将在之后AUTOSAR微控制器抽象层配置部分进行详细讲解。

```
    FUNC（void，ECUM_CODE）EcuM_AL_DriverInitZero（void）
{
    …
    Det_Init（）；
    …
}
```

```
FUNC（void，ECUM_CODE）EcuM_AL_DriverInitOne（const EcuM
_ConfigType* ConfigPtr）
{
    …
    Mcu_Init（ConfigPtr->ModuleInitPtrPB.McuInitConfigP
tr0_cpst）；
    McuFunc_InitializeClock（）；
    Port_Init（ConfigPtr->ModuleInitPtrPB.PortInitConfi
gPtr0_cpst）；
    Dio_Init（ConfigPtr->ModuleInitPtrPB.DioInitConfigP
tr0_cpst）；
    Gpt_Init（ConfigPtr->ModuleInitPtrPB.GptInitConfigP
tr0_cpst）；
    Adc_Init（ConfigPtr->ModuleInitPtrPB.AdcInitConfigP
tr0_cpst）；
    Can_Init（ConfigPtr->ModuleInitPtrPB.CanInitConfigP
tr0_cpst）；
    Icu_Init（ConfigPtr->ModuleInitPtrPB.IcuInitConfigP
tr0_cpst）；
    Pwm_Init（ConfigPtr->ModuleInitPtrPB.PwmInitConfigP
tr0_cpst）；
    …
}
```

（3）EcuMWakeupSources配置

在EcuMCommonConfiguration中需要配置EcuMWakeupModes，其配置如图6.40所示。

EcuMWakeupSources

	Short Name	EcuMValidationTimeout	EcuMWakeupSourceId
Filter			
0	ECUM_WKSOURCE_POWER	0.0	0
	ECUM_WKSOURCE_RESET	0.0	1
2	ECUM_WKSOURCE_INTERNAL_RESET	0.0	2
3	ECUM_WKSOURCE_INTERNAL_WDG	0.0	3
4	ECUM_WKSOURCE_EXTERNAL_WDG	0.0	4

图6.40　EcuMWakeupSources配置

（4）EcuMResetModes配置

本书示例EcuMResetModes配置如图6.41所示。

EcuMResetModes

	Short Name	EcuMResetModeId
Filter		
0	ECUM_RESET_MCU	0
	ECUM_RESET_WDGM	1
2	ECUM_RESET_IO	2

图6.41　EcuMResetModes配置

（5）EcuMShutdownCauses配置

本书示例EcuMShutdownCauses配置如图6.42所示。

图6.42　EcuMShutdownCauses配置

（6）EcuMDefaultAppMode配置

点击EcuMCommonConfiguration，本书示例选择EcuMDefaultAppMode为OSDEFAULTAPPMODE。见图6.43。

图6.43　EcuMDefaultAppMode配置

（7）EcuMGeneral配置

EcuMGeneral是EcuM模块整体功能的配置，本书示例配置EcuM_MainFunction（）函数调用周期为10ms。并且，需要添加自定义的头文件McuFunc.h。见图6.44。

图6.44　EcuMGeneral配置

6.5 BswM模块概念与配置方法介绍

BswM（Basic Software Mode Manager）模块属于系统服务层，它在系统服务层中的具体位置如图6.45所示。

图6.45　BswM模块具体位置

BswM模块有以下两大作用。

①模式仲裁（Mode Arbitration）：根据软件组件或其他BSW模块发来的模式请求（Mode Request）或者模式指示（Mode Indication），通过规则判断来发起相应模式切换。

②模式控制（Mode Control）：通过执行动作列表（Action List）里

面的动作实现模式切换。

模式仲裁和模式控制各司其职，模式仲裁以后如果确认要进行模式切换，那么模式控制就会执行在配置阶段预先定义好的这个模式切换需要执行的动作列表。

模式仲裁是基于规则（Rule）来进行的，每个规则中需要包含逻辑表达式（Logical Expression）和动作列表（Action List）。其中，逻辑表达式由一系列的模式请求条件（Mode Condition）通过逻辑运算符（AND/NAND/OR/XOR）组合起来；动作列表则由一系列动作（Actions）组合而成。

BswM模块相关概念示意如图6.46所示；BswM模块工作过程示意如图6.47所示。

图6.46　BswM模块相关概念示意

图6.47　BswM模块工作过程示意

在BswM的实现过程中，会在如下三种情况下进行模式仲裁：

①软件组件发出模式切换请求，即SWC通过调用与BswM模块交互的接口函数，BswM模式仲裁机制会根据当前已有的请求，以及当前请求的类型选择立即仲裁（IMMEDIATE）或者是推后仲裁（DEFERRED）；

②EcuM、WdgM等BSW模块的模式标识发生改变，进而向BswM模块发出一个模式仲裁请求；

③通过周期性地调用BswM_MainFunction（）函数，从而在BswM主函数中进行周期性的模式仲裁，该方式一般用来处理推后仲裁类型的请求。

模式控制最终是要执行动作列表中的动作，这些动作可以是调用其他BSW模块的服务或调用RTE、执行其他的动作列表或者引起另外的规则仲裁。

①条件执行（CONDITION）：每一次进行规则检查时，都会根据设定的期望结果来执行动作列表。

②触发执行（TRIGGER）：仅当当前规则检查结果和上次规则检查结果不同时才会执行动作列表。

为了简化BswM模块的设计，AUTOSAR规范中预定义了一些标准动作（Standard Actions）。

①ComM：设置一个通信接口的通信模式。

②ComM：限制通信模式。

③ComM：使能一个ComM通道的通信。

④LinSM：设置LIN调度表（Schedule Tables）。

⑤FlexRay：切换到“All Slot Mode”。

⑥Com：激活和停用I-PDU组（I-PDU Group），重设或不重设信号初始值。

⑦Com：使能或禁用截止时间超时监控（Deadline Timeout Monitoring）。

⑧Com：触发I-PDU发送。

⑨EcuM：设置ECU运行模式（ECU Operation Mode）。

⑩EcuM：关闭所有运行请求（Run Requests）。

⑪Network Management：使能或禁用网络管理通信（NM Communication）。

⑫PduR：使能或禁用PDU路由路径（PDU Routing Path）。

⑬RTE和BSW：调度器的模式切换。

本书示例中的BswM模块配置总览如图6.48所示。它主要完成一些BSW模块的初始化工作，并且还需要接收应用层EcuBaseSWC的启动/关闭CAN通信请求。

图6.48　BswM模块配置总览

BSW模块的初始化工作主要是基于EcuM等BSW模块的模式标识改变而向BswM模块发出模式仲裁请求的，发生在BSW模块间。为便于读者直观地感受BswM模块中各主要概念及其运行机制，这里以应用层EcuBaseSWC发起启动/关闭CAN通信请求为例进行介绍。

（1）BswMModeRequestPort配置

首先，需要建立一个与应用层SWC交互的服务端口，即添加一个BswMModeRequestPort，点击“+”Add，命名为BswM_MRP_ApplicationRequestPort，采用推后仲裁DEFERRED模式，如图6.49所示。

图6.49　BswMModeRequestPort新建

其次，还需要为该端口进行属性定义。由于该端口和应用层SWC交互，所以这里新建一个BswMGenericRequest，其请求类型RequestType为SWC，并且一共有两种模式（启动/关闭CAN通信），即配置BswM RequestedModeMax为2，如图6.50所示。

图6.50　BswMModeRequestPort→BswMGenericRequest配置

按照前述BswM模块的概念，下面需要定义ModeCondition、LogicalExpression、Action、ActionList、Rule以及BswMGeneral。

（2）ModeCondition与LogicalExpression配置

可按照与前述类似方法新建一个ModeCondition，命名为BswM_MC_AppReqFullCom，该条件为当BswM_MRP_ApplicationRequestPort端口模式请求值为1时为True，反之

为False，如图6.51所示。

图6.51　ModeCondition配置

定义了ModeCondition后，可以新建并配置LogicalExpression。这里新建名为BswM_LE_AppReqFullCom的LogicalExpression，并引用先前定义的BswM_MC_AppReqFullCom。由于只有一个条件，所以不需要逻辑运算符（Logical Operator），如图6.52所示。

图6.52　LogicalExpression配置

（3）Action与ActionList配置

在定义ActionList时需要定义Action，这里定义两个Action，分别为

BswM_AI_AppReqFullCom和BswM_AI_AppReqNoCom，它们分别使用ComM模块预定义的标准动作，请求COMM_NO_COMMUNICATION和COMM_FULL_COMMUNICATION，配置如图6.53所示。

图6.53　Action配置

之后，可定义ActionList。这里定义两个ActionList，分别为BswM_AL_AppReqFullCom和BswM_AL_AppReqNoCom，都配置成TRIGGER模式，并且分别引用BswM_AI_AppReqFullCom和BswM_AI_AppReqNoCom。Action List配置如图6.54所示。

图6.54　ActionList配置

（4）Rule配置

在定义了Mode Condition、Logical Expression、Action、Action List后，就可以定义Rule了。这里新建一个名为BswM_AR_AppReqFullNoCom的Rule，其配置如图6.55所示。

图6.55　Rule配置

（5）BswMGeneral配置

BswMGeneral配置主要是对BswM模块属性的配置，这里配置BswM_MainFunction（）周期为10ms，如图6.56所示。

图6.56　BswMGeneral配置

为了便于理解模式仲裁和模式控制的实现过程，这里先展现一下最终BswM模块生成的部分代码。其中，规则检查函数为BswM_Rule_BswM_AR_AppReqFullNoCom，这里以通过该规则检查为例进行分析。若规则检查结果为True，则调用BswM_ActionList_BswM_AL_AppReqFullCom函数，即触发相应ActionList；在ActionList函数中将调用Action函数BswM_Action_BswM_AI_AppReqFullCom。

```
FUNC（void，BSWM_CODE）BswM_Rule_BswM_AR_AppReqFul
```

```
lNoCom（void）
{
    if (BSWMLOGEXP_BSWM_LE_APPREQFULLCOM)
    {
        /* True Action list */
        /* Triggered */
        if（BswM_Prv_RuleState［6］！=BSWM_TRUE）
        {
          /* Make a call to the corresponding action lis
t item-BswM_AL_AppReqFullCom */
          BswM_ActionList_BswM_AL_AppReqFullCom（）；
      }
      /* Change the state of the rule-BswM_AR_AppReqFul
lNoCom */
  BswM_Prv_RuleState［6］=BSWM_TRUE;
    }
    else
    {
        /* False Action list */
        /* Triggered */
        if（BswM_Prv_RuleState［6］！=BSWM_FALSE）
        {
            /* Make a call to the corresponding action l
ist item-BswM_AL_AppReqNoCom */
            BswM_ActionList_BswM_AL_AppReqNoCom（）；
        }
        /* Change the state of the rule-BswM_AR_AppReqFu
llNoCom */
        BswM_Prv_RuleState［6］=BSWM_FALSE;
    }
```

```
}
FUNC（void，BSWM_CODE）BswM_ActionList_BswM_AL_AppReq
FullCom（void）
{
    VAR（Std_ReturnType，AUTOMATIC）action_RetVal_u8；
    /* Invoke all the available actions in the assendin
g order of ActionListItem index */
    /* Execute the Available Action-BswM_AI_AppReqFullC
om */
    BswM_Action_BswM_AI_AppReqFullCom（&action_RetVa
l_u8）；
    action_RetVal_u8=BSWM_NO_RET_VALUE；/* Avoid variab
le un-used compiler warning when only Rule or Action l
ist is configured as action */
}
FUNC（void，BSWM_CODE）BswM_Action_BswM_AI_AppReqFull
Com（P2VAR（Std_ReturnType，AUTOMATIC，BSWM_APPL_DATA）a
ction_RetVal_pu8）
{
    /* Initialize to"no return value" */
    *action_RetVal_pu8=BSWM_NO_RET_VALUE；
    /* Switch the communication mode for a ComM User-Co
mMUser_Can_Cluster_Channel */
    *action_RetVal_pu8=ComM_RequestComMode（0，COMM_
FULL_COMMUNICATION）；
}
```

6.6 BSW模块代码生成

在BCT界面中配置完所需要的BSW模块后，可以进行BSW模块相关代码与描述文件的生成，点击ISOLAR-A主菜单中“➡”右边箭头，选择Run Configuraions，如图6.57所示。将弹出如图6.58所示的界面。

图6.57　Run Configuraions配置（一）

图6.58　Run Configuraions配置（二）

在Run Configuraions界面中，在BSWGen Launch Configuraion菜单下新建一个名为LightECU_BSW的配置，即可进入BSW生成相关配置界面。其中，需要选择RTA-BSW工具版本、工程以及BSW代码生成路径，勾选需要生成的模块后，点击Run即可生成。

此时，Console界面中将显示生成情况信息，如图6.59所示。

图6.59　BSW模块生成过程

6.7 服务软件组件与应用层软件组件端口连接

在生成了BSW模块的代码后，切换到ISOLAR-A系统级设计界面，会发现产生一些基础软件模块的服务软件组件：

BswM、ComM、Det和EcuM等，如图6.60所示。

图6.60　生成了BSW后的服务软件组件

此时，如果涉及服务软件组件与应用层软件组件的交互，就需要为应用层软件组件添加相应的服务端口。本书示例中，在BswM模块配置阶段已经设计了相应的BswMModeRequestPort，名为BswM_MRP_ApplicationRequestPort，打开BswM软件组件也可以看到该

端口的信息，如图6.61所示。

图6.61　BswM软件组件

按照本书示例开发需求，该端口是和应用层EcuBaseSWC软件组件进行交互的，所以需要按照基于ISOLAR-A的软件组件设计方法中所提到的方法为EcuBaseSWC软件组件添加一个Port，其Port Interface为BswMSwcGenericRequest_ClientServerInterface，结果如图6.62所示。

图6.62　EcuBaseSWC软件组件

在完成了相关应用层软件组件的端口添加后，需要再次按照前述系统级设计与配置方法将这些新生成的服务软件组件都添加到VehicleComposition中，完成Assembly Connectors添加，并基于VehicleSystem完成SWC To ECU Mapping。最后，再次进行ECU Extract，此时的LightECU_FlatView Composition如图6.63所示。

图6.63　LightECU_FlatView Composition最终版

最终，生成EcuBaseSWC软件组件模板，并在其中调用它和BswM软件组件的模式请求RTE接口函数，编写相应模式切换逻辑即可。

6.8 RTE配置与代码生成

RTE生成器一般可工作在两个阶段：合同阶段（Contract Phase）和生成阶段（Generate Phase）。

6.8.1 RTE Contract阶段生成

RTE合同阶段（Contract Phase）的输入文件是软件组件描述文件，其输出文件为一些“.h”文件，主要包括类型定义和接口定义信息，如图6.64所示。其主要用于应用层软件开发阶段，此时完整的ECU定义可能还没完成，这样可以便于多方合作完成应用层软件的开发。根据项目开发情况，RTE Contract阶段生成时间较为灵活。

图6.64 RTE Contract阶段生成示意

点击ISOLAR-A主界面“R”RTE contract phase…，如图6.65所示，将弹出如图6.66所示界面。

图6.65 RTE Contract阶段生成（一）

图6.66　RTE Contract阶段生成（二）

选择RTE工具的路径、文件生成路径，加入Additional Commands，点击Finish即可完成RTE Contract阶段生成。

6.8.2　RTE配置

在生成RTE之前需要进行RTE配置。首先，需要完成ECU级模块配置信息的收集（Ecuc Value Collection）；其次，需要完成各运行实体到操作系统任务的映射（RE To Task Mapping）。

（1）ECU级模块配置信息收集

由于RTE是应用层与基础软件层交互的桥梁，所以在生成RTE之前需要收集所有ECU级模块的配置信息（Ecuc Value Collection）。右键点击Bsw文件夹
→Create BSW Module Descriptions→Create Ecuc Value Collections→Elem

如图6.67所示，将会弹出如图6.68所示的界面，和之前一样，需要创建一个AR Element。

图6.67　EcuC Value Collection新建（一）

图6.68　EcuC Value Collection新建（二）

双击LightECU_EcucValueCollection，会弹出如图6.69所示的界面，提示需要选择一个ECU Extract，选择EXTR_LightECU即可。

图6.69　EcuC Value Collection新建（三）

在新建了Ecuc Value Collection后，就可以向其中添加各模块的配置信息。右键点击LightECU_EcucValueCollection→New Child→Ecuc Values | Ecuc Module C即可新建一个引用，选择需要引用的模块配置信息即可。见图6.70。

图6.70　Ecuc Module Configuration Values Ref Conditional新建

由于RTE和OS两个模块的配置在BSW模块配置阶段还未添加，双击LightECU_EcucValueCollection，切换至RTE Configuration界面，点击Generate即可创建Rte模块；切换至OS Task Prroperties，点击Create OSAppMode即可创建Os模块。最终，本书示例ECU级模块配置信息收集结果如图6.71所示。

图6.71　Ecuc Value Collection结果

（2）运行实体到操作系统任务映射

由于运行实体是用户程序的最小划分，而操作系统任务是运行实体的载体。换言之，操作系统无法直接调度运行实体，而是需要通过将运行实体映射到不同的任务中，通过对任务进行调度进而实现各个特定的时刻执行特定的运行实体。所以，需要完成运行实体到操作系统任务的映射（RE To Task Mapping）。

这里需要映射的运行实体不仅包括应用层软件组件的，还包括基础软件模块的。最终，本书示例中运行实体到操作系统任务的映射关系设计如表6.2所示。其中，一共设计了7个任务来完成运行实体的调度。

表6.2　运行实体到操作系统任务的映射

Os Task/Event Mapping		Component Instance Properties	
OsTask	**OsPriority**	**Entities**	**ComponentInstance**
OsTask_ASW_10ms	5		
		RE_EcuBase_SWC	CPT_EcuBaseSWC
		RE_LightRequest	CPT_LightRequestSWC
OsTask_ASW_20ms	2		
		RE_LightControl	CPT_LightControlSWC
		RE_JudgeLightState	CPT_LightControlSWC
OsTask_BSW_10ms	15		
		BSWSE_EcuM_MainFunction	EcuM
		MainFunction	BSWIMPL_BswM
		BSWSE_ComM_MainFunction_0	BSWIMPL_ComM
		BSWSE_Com_MainFunctionRx	BSWIMPL_Com
		BSWSE_Com_MainFunctionTx	BSWIMPL_Com
		BSWSE_MainFunction	BSWIMPL_CanSM
		BSWSE_MainFunction_BusOff	BSWIMPL_Can
OsTask_BSW_1ms	30		
		BSWSE_MainFunction_Read	BSWIMPL_Can
		BSWSE_MainFunction_Write	BSWIMPL_Can
		RE_SetEcuAlive1	CPT_EcuAliveIndicatorSWC
OsTask_BSW_2ms	25		
		RE_SetEcuAlive2	CPT_EcuAliveIndicatorSWC
OsTask_BSW_5ms	20		
		RE_SetEcuAlive3	CPT_EcuAliveIndicatorSWC
OsTask_Init	40		
		LightRequest_Init	CPT_LightRequestSWC
		LightControl_Init	CPT_LightControlSWC

根据表6.2，可以完成运行实体到操作系统任务的映射工作。双击LightECU_EcucValueCollection，切换至Entity to Task Mapping界面。右键点击可新建Os Task，如图6.72所示。

图6.72　OsTask新建

在创建了上述所有Task后，切换到Os Task Priorities界面可以看到所有的Task，并可以为其配置相关属性，如图6.73所示。

图6.73　Os Task Priorities界面

完成运行实体到操作系统任务的映射时，只需要在Entity to Task Mapping界面将右边UnMapped Entities中的运行实体拖到左边Mapped Entities中对应的Task即可，工具将自动完成映射，如图6.74所示。

图6.74　Entity to Task Mapping

6.8.3　RTE Generation阶段生成

在完成了RTE配置后，就可以进行RTE生成阶段（Generation Phase）的生成，该阶段将主要生成如下文件，如图6.75所示。

图6.75　RTE Generation阶段生成示意

①Rte_Type.h：通用Type定义。

②Rte_<SWC Name>.h：应用API声明。

③Rte_<SWC Name>_Type.h：软件组件特定的Type定义。

④Rte.c：API定义、数据结构定义。

⑤Rte_Lib.c：静态RTE库函数。

⑥<Task Name>.c：Task主体函数。

⑦osNeeds.arxml：OS配置描述。

由于RTE是应用层软件和基础软件的桥梁，所以一般

RTE　Generation阶段在ECU级开发末期生成，即此时已完成了基本所有的SWC与BSW模块的设计。

点击ISOLAR-A主界面“R”Generate RTE code in Generate phase，如图6.76所示，将弹出如图6.77所示界面。

图6.76　RTE Generation阶段生成（一）

图6.77　RTE Generation阶段生成（二）

选择RTA-RTE工具的路径、文件生成路径，并选择一个ECU Instance，加入Additional Commands，点击Finish即可完成RTE Generation阶段生成。

由于RTA-RTE在RTE Generation阶段将生成部分操作系统OS的配置，它们可以导入RTA-OS工具，简化OS的配置过程，所以在OS配置生成前先完成RTE Generation阶段生成。下面展示本书示例RTE Generation阶段生成的一些代码。

首先，展示的是LightControlSWC软件组件向外发送车灯当前状态的RTE接口实现，可见在该函数中直接调用标准接口Com_SendSignal（）。

```
    FUNC（Std_ReturnType，RTE_CODE）
Rte_ImplWrite_LightControlSWC_PPortSendLightState_D
ESendLightState（VAR（UInt8，AUTOMATIC）data）/* 1 */
{
  VAR（Std_ReturnType，AUTOMATIC）rtn=RTE_E_OK；

  /* SpecReq：Send signal begin */
  /* The signal is LightState */
  if（（（VAR（StatusType，AUTOMATIC））E_OK）！=Com_Send
Signal（（（VAR（Com_SignalIdType，AUTOMATIC））1），&dat
a））
  {
      rtn=（（VAR（Std_ReturnType，AUTOMATIC））RTE_E_COM
_STOPPED）；
  }
  /* SpecReq：Send signal end */
  /* Send complete */
  return rtn；
}
```

其次，展现的是Client/Server模式的具体实现。由于本书示例采用

了同步模式，所以当Client端调用接口Rte_Call_LightControlSWC_RPortGetLightState_OPGetLightState时，即调用了Server端名为RE_GetLightState的函数。

```
    FUNC（Std_ReturnType，RTE_CODE）
Rte_Call_LightControlSWC_RPortGetLightState_OPGetLightState（CONSTP2VAR（UInt8，AUTOMATIC，RTE_APPL_DATA）DEGetLightState）/* 1 */
{
  VAR（Std_ReturnType，AUTOMATIC）rtn；

  /* SpecReq：Activate RE via Queue begin */
  /* Parameter DEGetLightState has direction OUT */
  RE_GetLightState（DEGetLightState）；
  rtn=（（VAR（Std_ReturnType，AUTOMATIC））RTE_E_OK）；
  /* SpecReq：Activate RE via Queue end */
  return rtn；
}
```

最后，展示一下Task主体函数的代码，可见在名为OsTask_BSW_1ms的Task主体函数中调用了先前映射到它上面的三个运行实体函数。

```
    TASK（OsTask_BSW_1ms）
{
  /* Box：Implicit Buffer Initialization begin */
  /* Box：Implicit Buffer Initialization end */
     /* Box：Implicit Buffer Fill begin */
  /* Box：Implicit Buffer Fill end */
```

```
  {
    /* Box: BSWImpl6_BSWIMPL_Can begin */
    Can_MainFunction_Read ();
    /* Box: BSWImpl6_BSWIMPL_Can end */
  }
  {
    /* Box: BSWImpl6_BSWIMPL_Can begin */
    Can_MainFunction_Write ();
    /* Box: BSWImpl6_BSWIMPL_Can end */
  }
  {
    /* Box: CPT_EcuAliveIndicatorSWC begin */
    SetEcuAlive1 ();
    /* Box: CPT_EcuAliveIndicatorSWC end */
  }
  /* Box: Implicit Buffer Flush begin */
  /* Box: Implicit Buffer Flush end */
  TerminateTask();
} /* OsTask_BSW_1ms */
```

6.9 AUTOSAR操作系统概念与配置方法介绍

6.9.1 AUTOSAR操作系统概念

AUTOSAR操作系统（Operating System，OS）属于系统服务层，它是一种多任务的实时操作系统（Real Time Operating System，RTOS），并且是静态操作系统，即不可以在运行时动态创建任务。实时操作系统对系统的实时性要求较高，需要保证在特定的时间内处理完相应的事件或数据。AUTOSAR操作系统源于OSEK/VDX操作系统，不仅继承了OSEK操作系统高效、可靠的优点，而且还对其进行了功能拓展。下面对AUTOSAR操作系统的相关概念进行介绍。

（1）操作系统的任务（Task）及其状态

操作系统的任务（Task）主要可分为用户任务和系统任务。用户任务需要根据不同应用场景进行自定义。用户任务的划分及其优先级的选取是操作系统配置的重点，这在很大程度上影响着程序的执行效率和结果。而系统任务一般有空闲任务（Idle Task），它的优先级最低，并且它的执行时间是衡量CPU负载率的重要参数。所以，操作系统的任务设计一般指的是用户任务的设计，AUTOSAR OS定义了两种用户任务：

①基本任务（Basic Task）；

②扩展任务（Extended Task）。

其中，基本任务状态包括运行状态（Running）、就绪状态（Ready）和挂起状态（Suspended），任务切换只发生在这3种状态之间；而扩展任务除了具有基本任务的3种状态外，还有等待状态（Wating），如图6.78所示。

图6.78　基本任务（左）与扩展任务（右）状态模型

对于上面提及的任务状态，它们分别有如下特征。

①运行状态（Running）：处于运行状态时，处理器资源被分配给该任务，该任务的指令被执行。在同一个处理器上任何时候只有一个任务能处于运行状态，而处于其他状态的任务则可以有多个。

②就绪状态（Ready）：处于就绪状态是任务转换到运行状态的前提，此时任务等待处理器的资源分配，由调度器来决定哪个就绪任务将被执行。

③挂起状态（Suspended）：处于挂起状态时，任务是被动的，可以被激活。

④等待状态（Wating）：任务因等待一个或多个事件而无法继续执行。

基本任务只有在以下情况才会释放处理器资源。

①情况1：该任务运行结束时。

②情况2：操作系统切换到更高优先级的任务时。

③情况3：发生了一个中断，处理器切换到该中断对应的中断服务程序时。

基本任务的代码如下。

```
    #include“OS.h”
TASK（BasicTask）
{
    ...
    /* User code */
    ...
    TerminateTask（）；
}
```

扩展任务在运行状态通过调用WaitEvent（）函数切换为等待状态，直到所等待的事件发生。扩展任务在等待状态下会释放处理器资源，操作系统会执行处于就绪状态且任务优先级最高的任务，而不需要终止该扩展任务。所以，扩展任务比基本任务要更复杂，需要占用更多的系统资源。扩展任务的代码如下。

```
    #include“OS.h”
TASK（ExtendedTask）
{
  for（；；）
  {
       WaitEvent（Event1）；
    /* perform actions */
    ClearEvent（Event1）；
  }
}
```

（2）任务的调度策略（Scheduling Policy）

AUTOSAR　OS的任务调度是基于优先级（Priority）的，每个任务都要根据其特性预先定义一个优先级，并且需要配置其可抢占属性，可

抢占属性分为非抢占与全抢占，这里的抢占指的是内核可抢占。根据操作系统中各任务的可抢占性配置情况，AUTOSAR　OS可提供以下三种调度策略（Scheduling Policy）。

①非抢占式（Non-preemptive）：操作系统中所有任务都被定义成不可抢占的。

②完全抢占式（Preemptive）：操作系统中所有任务都被定义成可抢占的。

③混合抢占式：操作系统中有的任务被定义成可抢占的，而有的任务则被定义成不可抢占的。

若采用全抢占式任务调度策略，运行的任务在任何时刻都有可能由于更高优先级的任务就绪而被迫释放处理器，此时具有最高优先级的就绪任务将被调度运行，而当前低优先级的任务就将从运行状态切换到就绪状态，并将当前运行环境保存下来，待下次继续运行时恢复。全抢占式任务调度策略示意如图6.79所示，其中Task A优先级高于Task B，当激活Task A后进入就绪状态，Task B则被迫释放处理器，调度程序开始调度Task　A从就绪状态切换到运行状态，直至Task　A完成后，再调度Task B继续运行。

图6.79　全抢占式任务调度策略

若采用非抢占式任务调度策略，任务在执行期间不会被高优先级的任务抢占，任务的切换只发生在当前任务完成时，它的最大的缺点是任务响应时间不确定，导致系统实时性较差。非抢占式任务调度策略如图6.80所示，可见虽然具有高优先级的任务Task A被激活而切换到就绪状态，但还是得等到低优先级的任务Task B运行结束才被调度。

图6.80　非抢占式任务调度策略

若采用混合抢占式任务调度策略，则操作系统的调度策略取决于当前任务的可抢占属性。如果当前任务为非抢占类型任务，则操作系统采用非抢占式任务调度策略；反之，如果当前任务为全抢占类型任务，则操作系统采用全抢占式任务调度策略。

（3）计数器（Counter）与报警器（Alarm）

AUTOSAR OS提供了处理重复事件的服务。它是基于计数器（Counter）与报警器（Alarm）来实现的。反复出现的事件由特定的计数器来记录。在计数器的基础上，AUTOSAR OS向应用软件提供了报警机制。多个报警器可以连接到同一个计数器。当到达报警器相对应的计数器设定值时，可激活一个任务、设置一个事件或调用一个回调函数。这里的计数器设定值可以定义为实际计数器值的绝对值（绝对报警器）或者是相对值（相对报警器）。

（4）调度表（Schedule Table）

使用一个计数器和一个基于该计数器的报警器队列可以实现静态定义的任务激活机制，当计数器的计数值依次到达报警设定值时，各报警器被触发。但这样很难保证各报警器之间具有特定的时间间隔；并且由于每个报警器只能激活一个任务或者设置一个事件，所以需要定义多个报警器来实现在同一时刻激活多个任务或者设置多个事件。为解决上述问题，AUTOSAR OS引入了调度表（Schedule Table）的概念。

调度表中可以定义一系列终结点（Expiry Point）。每个调度表都有一个以Tick为单位的持续时间（Duration）。其中，每个终结点都有一个以Tick为单位的距离调度表起始点的偏移量（Offset）。在每个终结点可以进行一个乃至多个激活任务或者设置事件的操作。

与报警器类似，一个调度表由一个计数器驱动。调度表有以下两种运行模式。

①单次执行（Single shot）：调度表启动后只运行一次，并在调度表的终点自动停止。此时，每个终结点只处理一次。

②重复执行（Repeating）：调度表启动后可重复运行，即当到达调度表终点时，又再次回到起点重复运行。此时，每个终结点将以调度表的持续时间为周期，周期性地被处理。

（5）中断（Interrupt）处理

在AUTOSAR中定义了两类中断服务程序（Interrupt Service Routine，ISR）。

①一类中断（Category 1 Interrupt）：此类中断服务程序不能使用操作系统的服务，中断服务程序结束后，处理程序将从产生中断的地方继续执行。由于该类中断不影响任务的管理，所以占用的系统资源较少。

②二类中断（Category 2 Interrupt）：此类中断服务程序可以使用一部分操作系统提供的服务，如激活任务、设置事件等。

在AUTOSAR　OS中，任务的优先级低于中断的优先级，即最低优先级的中断可以打断最高优先级的任务。所以，中断服务程序的执行时间不易太长，以免耽误重要任务的执行，从而降低整个操作系统的实时性。

（6）资源管理

资源管理被用来协调有着不同优先级的多个任务对共享资源（如内存或硬件等）的并发访问（Concurrent Access）。

AUTOSAR操作系统采用优先级上限协议（Priority Ceiling Protocol）来避免优先级倒置（Priority Inversion）和死锁（Deadlock）问题的发生。在系统初始化阶段，每个资源拥有的上限优先级是静态分配的，资源的上限优先级必须高于所有要访问该资源的任务和中断的最高优先级，但是低于不访问该资源的任务的最低优先级。

如果一个任务需要访问一个资源，并且该任务的优先级比该资源的优先级上限低，则将该任务的优先级提升到所要访问的资源的上限优先级；当该任务释放资源后，其优先级再回到要求访问该资源前的优先级。

（7）自旋锁（Spin Lock）

自旋锁（Spin Lock）是一种为保护共享资源而提出的锁机制，一般用于多核处理器解决资源互斥问题。当内核控制路径必须访问共享数据结构或进入临界区时，如果自旋锁已经被别的执行单元保持，调用者就一直循环在那里看是否该自旋锁的保持者已经释放了该锁，从而起到对某项资源的互斥使用。

（8）一致类（Conformance Class）

AUTOSAR OS支持4种一致类（Conformance Class），可使得开发者根据实际应用需求灵活地配置操作系统的调度程序。一致类的划分是根据每个优先级可能具有的任务个数、需要的是基本任务还是扩展任务等来决定的。按照大类可分为基础一致类（Basic Conformance Class，BCC）和扩展一致类（Extended Conformance Class，ECC），每个大类又可分为两个小类，具体如下。

①BCC1：每个优先级只有一个任务，基本任务的激活数只能为一次，仅支持基本任务。

②BCC2：每个优先级可有多个任务，基本任务的激活数可为多次，仅支持基本任务。

③ECC1：每个优先级只有一个任务，基本任务的激活数只能为一次，支持基本任务和扩展任务。

④ECC2：每个优先级可有多个任务，基本任务的激活数可为多次，支持基本任务和扩展任务。

（9）可扩展性等级（Scalability Class）

为了迎合不同用户对操作系统功能的不同需求，AUTOSAR OS可以根据其可扩展性等级（Scalability Class，SC）分为以下四类。

①SC1：在OSEK OS基础上加入调度表（Schedule Table）。

②SC2：在SC1基础上加入时间保护（Timing Protection）。

③SC3：在SC1基础上加入存储保护（Memory Protection）。

④SC4：在SC1基础上加入时间保护和存储保护。

6.9.2 RTA-OS工程创建

在进行操作系统OS配置之前需要新建一个RTA-OS工程，点击RTA-OS主菜单File→New Project，如图6.81所示。之后，将会弹出如图6.82所示界面。

图6.81　RTA-OS工程新建（一）

图6.82　RTA-OS工程新建（二）

由于配置过程本质是对配置描述文件的修改过程，所以在新建工程阶段需要完成XML Settings，即对其中一些元素进行命名，依次为AR Package Name、ECU Configuration Name、OS Configuration Name、Release。输入相应名字，点击OK即可完成工程新建。

此时，在RTA-OS主界面的OS Configuration界面中将会显示新工程

的所有配置项，如图6.83所示。其中，大部分配置项的概念在前面AUTOSAR OS基本概念介绍过程中已经涉及。

图6.83　RTA-OS工程新建结果

下面先对本书示例中所使用的OS进行一些整体性的配置，即完成General中的配置，部分配置如图6.84所示。本书示例使用SC1等级的OS，并使能Startup、Shutdown与Error钩子（Hook）函数，便于OS的调试。

图6.84　OS General配置

其次，需要选择OS所运行的目标芯片，本书示例所用硬件平台为MPC5744P芯片，如图6.85所示。

图6.85　OS General→Target配置

6.9.3　AUTOSAR操作系统配置方法

由于AUTOSAR中提出了运行时环境（RTE）的概念，先前也已经完成了运行实体RE向操作系统任务Task的映射，并完成了RTE　Generation阶段的生成。其中，RTE生成的osNeeds.arxml描述文件与先前在RTE配置阶段创建的OsCfg.arxml描述文件中包含了与操作系统相关的配置信息，将这两个描述文件直接导入刚才创建的RTA-OS工程即可完成OS中与用户任务相关的大部分配置。下面介绍基于RTE的OS配置方法。

（1）描述文件导入

切换到Project　Files菜单，右键点击OS工程文件LightECU.rtaos→Add　Existing　File，如图6.86所示。选择添加先前所述的OsNeeds.arxml与OsCfg.arxml文件即可。

图6.86　OS配置文件导入（一）

描述文件导入成功后，如图6.87所示，其中粗体的文件为当前正在被修改的描述文件，本书示例工程基于RTA-OS工程新建时产生的描述文件进行修改。

图6.87　OS配置文件导入（二）

此时，再切换到OS Configuration界面，将看到大部分与用户任务相关的配置信息已导入进来，如图6.88所示。

图6.88　OS配置信息导入结果

（2）Counter配置

由于在ISOLAR-A中进行RTE配置时未定义OS　Counter具体实现相关的属性，所以需要进行添加。本书示例中将Counter的计数基准配置为1ms，即将Rte_TickCounter中的Seconds　Per　Tick设置为0.001，并将

Ticks Per Base设置为1，如图6.89所示。

图6.89　OS Counter配置

（3）ISR配置

由于ISR是与具体实现密切相关的部分，而且还与微控制器相关，所以也需要另行配置。可以右键点击ISRs→New新建一个中断服务函数，其名字即为中断服务函数名。这里以产生OS　　Tick的通用定时器（GPT）中断配置为例进行说明。这里新建一个名为Gpt_STM_0_Ch_0_ISR的ISR，将其配置为二类中断（CATEGORY_2），定义一个优先级（Priority），并需要选择一个中断向量号（Address/Vector），本书例程采用STM_0_CH_0来实现GPT，所以选择System Timer 0 Channel 0（INTC_36）即可，如图6.90所示。

图6.90　ISR配置

除了产生OS Tick的GPT中断，根据微控制器抽象层中一些模块及硬件通道的选择还需要完成一些ISR的配置。在本书示例中，对于A型车灯而言，还需要配置模数转换（ADC）模块的ADC_EOC中断，其中断函数名为Adc_Adcdig_EndGroupConvUnit0；对于B型车灯则还需要配置输入捕获（ICU）相关的中断，其中断函数名为ETIMER_2_CH_4_ISR，它们的配置方法与上述GPT中断配置方法一致。

（4）Schedule Table配置

本书示例是利用一个Schedule Table来实现各Task调度的，自动生成的配置如图6.91所示。该Schedule Table的驱动计数器为Rte_TickCounter，定义成了重复模式，且其持续时间为20ms。Schedule Table中一共定义了20个终结点，在每个终结点定义了若干操作。

图6.91　Schedule Table配置

（5）Task配置

在导入的Task配置信息的基础上，还需要新建一个初始化任务，它具有最高的优先级，并且只运行一次，在OS启动之后首先被调用，完成一些初始化的工作，如调用EcuM_StartupTwo（）函数完成一些BSW模块的初始化。

右键点击Tasks→New新建一个Task，将其命名为ECU_StartupTask。在General配置菜单中可配置该Task的基本属性。这里将该Task配置为所有Task中的最高优先级，由于数值越大，优先级越高，这里Priority选为50；将激活方式（Activation）配置为1，表示该任务在任何时候只允许激活一次；可抢占属性（Task Preemptability）配置为NON，即该Task配置为非抢占式。其次，需要切换到Resource界面，添加所有与BSW模块相关的资源；最后，切换到Application Modes界面，添加一个Application Mode OSDEFAULTAPPMODE，即将ECU_StartupTask配置为在OSDEFAULTAPPMODE模式下自启动的Task，即在OS启动后自动启动该Task。ECU_StartupTask配置如图6.92所示。

图6.92　ECU_StartupTask配置

最终，本书示例中OS的Task配置如图6.93所示。

OS Configuration > Tasks

There are currently 8 tasks.

	Name	Priority	Task Preemptability	Activations	Application	
▶	ECU_StartupTask	50	NON	1	<undefined>	...
	OsTask_ASW_10ms	5	FULL	1	<undefined>	...
	OsTask_ASW_20ms	2	FULL	1	<undefined>	...
	OsTask_BSW_10ms	15	FULL	1	<undefined>	...
	OsTask_BSW_1ms	30	FULL	1	<undefined>	...
	OsTask_BSW_2ms	25	FULL	1	<undefined>	...
	OsTask_BSW_5ms	20	FULL	1	<undefined>	...
	OsTask_Init	40	NON	1	<undefined>	...
*						

图6.93　OS Task配置

6.9.4　RTA-OS工程编译

RTA-OS工具可以直接调用编译器对OS相关代码进行编译。在完成了OS所有配置后，可以切换到Builder→Setup界面，对生成文件的路径、包含的头文件等进行设置后，工具会自动生成Build脚本，如图6.94所示。

图6.94　Builder→Setup配置

配置完成后，切换到Builder→Build界面，点击Build　Now按钮即可开始OS工程的编译，如图6.95所示。过程中RTA-OS将调用编译器完成OS相关代码的编译。

图6.95　RTA-OS工程的编译

6.10 本章小结

本章在介绍ETAS针对AUTOSAR运行时环境和基础软件层开发工具RTA-RTE、RTA-BSW和RTA-OS的基础上，对ECU级开发中除微控制器抽象层MCAL以外部分进行了较为全面的讲解。在讲解过程中，从各模块概念着手，结合工具配置方法和一些生成的代码进行剖析。通过本章的学习，可以对BSW（除MCAL外）各常用模块和RTE的作用、配置方法有一个较为全面的认识。

第7章　AUTOSAR ECU级开发之MCAL

微控制器抽象层（Microcontroller Abstraction Layer，MCAL）位于AUTOSAR软件架构的最底层，与微控制器的内部单元及其外设相关，接收上层指令，完成对硬件的直接操作；并获取硬件相关状态，反馈给上层，对上层屏蔽了硬件相关特征，只提供对应的操作接口。

在AUTOSAR方法论中，AUTOSAR微控制器抽象层配置与实现属于ECU级开发范畴，由于MCAL模块较多，并且需要使用MCAL配置工具完成配置，所以这里单独成章专门介绍AUTOSAR微控制器抽象层配置与实现方法。

本书示例中需要使用到MCAL中微控制器驱动中的MCU驱动（Microcontroller Unit Driver）、GPT驱动（General Purpose Timer Driver）；I/O驱动中的PORT驱动、DIO驱动（Digital Input/Output Driver）、ADC驱动（Analog-to-Digital Converter Driver）、PWM驱动（Pulse Width Modulation Driver）以及ICU驱动（Input Capture Unit Driver）；通信驱动中的CAN驱动（CAN Driver）。

7.1 MCAL配置工具入门

恩智浦（NXP）公司与Elektrobit（EB）公司合作开发了针对MPC5744P芯片微控制器抽象层（MCAL）的配置工具及代码。其中，配置工具是基于EB tresos Studio平台的，MCAL代码则由NXP提供。

7.1.1 MCAL配置工具安装方法

（1）EB tresos Studio工具安装方法

双击EB tresos Studio工具安装包里面的setup.exe即可弹出如图7.1所示的界面。其中，左边菜单栏可以选择需要安装的项目，右侧则可以选择安装路径，确认上述信息后，点击Install即可安装EB tresos Studio工具。

图7.1　EB tresos Studio安装

（2）MPC5744P MCAL代码包安装方法

双击MPC5744P MCAL代码包中的MPC574XP_MCAL4_0_RTM_2_0_1.exe文件，按照默认操作安装即可。见图7.2。

图7.2　MPC5744P MCAL代码包安装（一）

MPC5744P MCAL代码包安装成功之后，需要将其中plugins文件夹下面的各模块相关文件夹复制到EB tresos Studio工具安装路径下的plugins同名文件夹内，如图7.3所示。这样就完成了MCAL配置环境的搭建。

图7.3　MPC5744P MCAL代码包安装（二）

7.1.2　MCAL配置工具界面说明

EB tresos Studio工具界面如图7.4所示，由于它也是基于Eclipse平台开发的，所以整体界面风格与ISOLAR-A类似。左侧Project　Explorer可进行MCAL工程浏览；选择特定模块后，可在中间配置界面进行相关配置；点击每个配置项，可在右下角Description窗口中查看配置项说明及配置方法提示。

图7.4　EB tresos Studio工具界面

7.1.3　MCAL配置工程创建方法

打开EB tresos Studio工具，点击File→New→Configuration Project，如图7.5所示。将会弹出如图7.6所示的New Configuration Project界面。

图7.5　新建MCAL配置工程（一）

图7.6　新建MCAL配置工程（二）

在New Configuration Project界面中，输入Project Name、选择工程路径、勾选AUTOSAR版本之后，点击Next，将会进入如图7.7所示的Configuration Project Data界面。

图7.7　新建MCAL配置工程（三）

在Configuration Project Data界面中，输入ECU ID，并选择Target，即微控制器型号之后，点击Next将进入Module　Configurations界面，如图7.8所示。

图7.8　新建MCAL配置工程（四）

在Module Configurations界面左侧Available Modules菜单栏中可以看到一些非MCAL的模块，如CanIf、EcuM等，这里都不作使用，因为除MCAL外的其余BSW模块基于前述AUTOSAR系统解决方案，是通过ETAS RTA-BSW和RTA-DS来进行配置生成的。在左侧Available Modules菜单栏中选择所需要添加的MCAL模块，点击中间Add module configurations for selected modules图标则可以完成MCAL需求模块的添加。最后，点击Finish即可完成工程创建，MCAL配置工程创建结果如图7.9所示。

图7.9　MCAL配置工程创建结果

7.2 MCAL模块配置方法及常用接口函数介绍

7.2.1 Mcu模块

MCU驱动（Microcontroller Unit Driver）位于微控制器抽象层，它可以直接访问微控制器的硬件。MCU驱动提供微控制器的初始化、复位、休眠等功能，还可以为其他MCAL模块提供所需要的与微控制器相关的特殊功能。

MCU驱动可以使能MCU时钟，并设置MCU时钟相关的参数，例如：CPU时钟、锁相环（Phase Locked Loop，PLL）、外设时钟、预分频器等的参数。若系统需要进入低功耗模式，MCU驱动还需完成微控制器在各种状态之间转换所需的工作。

（1）McuGeneral配置

McuGeneral配置主要是对Mcu模块整体功能的配置，如图7.10所示。其中，主要配置项如下。

图7.10　McuGeneral配置

①Mcu Development Error Detect：Mcu模块开发错误检测使能。

②Mcu Get Ram State API：获取RAM状态API使能。

③Mcu Init Clock API：初始化时钟API使能。

④Mcu No PLL：锁相环禁用。

⑤Mcu Enter Low-Power Mode：进入低功耗模式使能。

⑥Mcu Perform Reset API：执行复位API使能等。

（2）McuResetReasonConf配置

在ECU中，有很多不同的原因可以造成MCU复位，如果硬件允许，Mcu模块可以获取复位的原因。McuResetReasonConf配置可以添加不同的Mcu复位原因，点击“+”Add new element with default values添加即

可，如图7.11所示。

图7.11　McuResetReasonConf配置

（3）McuModuleConfiguration配置

McuModuleConfiguration配置较为复杂，这里择其重点进行讲解。

McuModuleConfiguration→General配置（图7.12）中External Crystal Frequency为外部晶振频率，本书使用的硬件平台为MPC5744P开发板，外部晶振频率为40MHz。

图7.12　McuModuleConfiguration→General配置

McuModuleConfiguration→McuClockSettingConfig主要是单片机时钟的配置，是Mcu模块配置的重点与难点，这里可以参考MPC5744P芯片手册的Clock　generation图片（图7.13），通过该图片可以理清各时钟间的关系，从而有助于Mcu模块的相关配置。

图7.13　MPC5744P Clock generation

点击“+”Add new element with default values可添加McuClockSettingConfig配置，如图7.14所示。双击McuClockSettingConfig的Index可进入如图7.15所示的配置界面。

图7.14　McuModuleConfiguration→McuClockSettingConfig配置

图7.15　McuModuleConfiguration→McuClockSettingConfig→General配置

McuClockSettingConfig模块主要是对MPC5744P Clock generation图中的各模块进行配置。

在McuClockSettingConfig→General配置界面主要需要进行如下配置。

①System Clock Frequency：系统时钟频率，这里配置成200MHz。

②System Clock Select：系统时钟源选择，可以选择内部晶振（IRC）、外部晶振（XOSC）、锁相环0（PLL0_PHI）、锁相环1（PLL1_PHI），这里选择PLL0_PHI。

由于本书示例使用PLL0_PHI，所以可以切换到McuPll_0进行锁相环0的相关配置，如图7.16所示。

图7.16　McuModuleConfiguration→McuClockSettingConfig→McuPll_0配置

其中，一些参数需要通过如下计算公式计算。

①PLL_0_PHI_Frequency=（McuAuxClk3_Frequency *McuLoopMulti
（McuInputClkPreDivider *McuPHI_ReducedFreqDivider）。

②PLL_0_PHI1_Frequency=（McuAuxClk3_Frequency *McuLoopMul
（McuInputClkPreDivider *McuPHI1_ReducedFreqDivider）。

③Fvco=（2*McuAuxClk3_Frequency *McuLoopMultiplicationClkDivi

在配置了系统时钟与锁相环后，就需要对MPC5744P Clock generation图中右侧通过分频后的输出时钟进行配置，如ADC_CLK、PBRIDGE0_CLK等。可以先定义McuClockReferencePoint，即对项目中使用到的时钟进行全局的配置，从而可将外设时钟通过McuClockReferencePoint与其他BSW模块联系起来，如图7.17所示。

图7.17　McuModuleConfiguration→McuClockSettingConfig→McuClockReferencePoint配置

之后，就需要对上述各时钟涉及的AUX Clock Selector进行配置。下面以AUX Clock Selector0为例，其时钟源为锁相环0（PLL0_PHI），通过它可分频给MOTC_CLK、SGEN_CLK和ADC_CLK。如图7.18所示，McuAuxiliaryClock0Divider0为分频器0，是给MOTC_CLK用的。

图7.18　McuModuleConfiguration→McuClockSettingConfig→McuAuxClock0配置

其中，主要配置项说明如下。

①Auxiliary Clock0 Divider0 Frequency：分频后的频率，此处MOTC_CLK需要50MHz，与McuClockReferencePoint_MC_CLK对应。

②Auxiliary Clock0 Divider0 Enable：分频使能。

③Auxiliary Clock0 Divisor0：分频系数，此处由PLL0_PHI的200MHz分频到50MHz，所以分频系数为200MHz/50MHz=4。

（4）Mcu模块初始化相关函数介绍

当Mcu模块配置完成后，在实际使用中，需要对Mcu模块进行初始化。由于Mcu模块是系统正常工作的基础，所以它的初始化软为靠前。并且，其初始化过程需要按照如下流程进行初始化相关函数的调用：

Mcu_Init；

Mcu_InitClock；

Mcu_GetPllStatus（若使用PLL，直到PLL锁相环配置成功）；

Mcu_DistributePllClock（若使用PLL）；

Mcu_InitRamSection（按照具体需求，可不调用）。

上述函数的具体定义如下。

①函数Mcu_Init：

void Mcu_Init（const Mcu_ConfigType *ConfigPtr）；

参数：const Mcu_ConfigType *。

返回值：void。

②函数Mcu_InitClock：

Std_ReturnType Mcu_InitClock（Mcu_ClockType ClockSetting）；

参数：Mcu_ClockType。

返回值：Std_ReturnType。成功：E_OK。不成功：E_NOT_OK。

③函数Mcu_GetPllStatus：

Mcu_PllStatusType Mcu_GetPllStatus（void）；

参数：void。

返回值：Mcu_PllStatusType。有如下3种情况：
MCU_PLL_STATUS_UNDEFINED表示PLL状态未知；
MCU_PLL_LOCKED表示锁相环配置已经成功；
MCU_PLL_UNLOCKED表示锁相环配置还未成功。

④函数Mcu_DistributePllClock：

void Mcu_DistributePllClock（void）；

参数：void。

返回值：void。

⑤函数Mcu_InitRamSection：

Std_ReturnType Mcu_InitRamSection（Mcu_RamSectionType RamSec

参数：Mcu_RamSectionType。

返回值：Std_ReturnType。成功：E_OK。不成功：E_NOT_OK。

本书示例中，使用了PLL，其Mcu模块的初始化过程代码如下。

```
    Mcu_Init（&McuConfigPB_0）；
void McuFunc_InitializeClock（void）
    {
    Mcu_InitClock（McuConf_McuClockSettingConfig_McuClo
ckSettingConfig）；
```

```
    while（MCU_PLL_LOCKED! =Mcu_GetPllStatus（））
    {
    }
    Mcu_DistributePllClock（）；
}
```

7.2.2 Gpt模块

GPT驱动（General Purpose Timer Driver）使用通用定时器单元的硬件定时器通道，为操作系统或者其他基础软件模块提供计时功能。GPT驱动可以提供启动和停止硬件定时器、得到定时器数值、控制时间触发的中断、控制时间触发的中断唤醒等功能。GPT通道可以设置为连续模式（CONTINUOUS）或单次模式（ONESHOT）。

①连续模式：定时器到达目标时间会自动清零并继续运行。

②单次模式：定时器到达目标时间，即计数值达到设定值时，定时器会自动停止，保持计数值不变，且通道状态从“运行”变为“超时”。

本书示例中使用GPT为操作系统提供计时功能，所以需要配置一个GPT通道。

（1）Gpt General配置

Gpt General配置主要是对Gpt模块整体功能的配置，如图7.19所示。其中，主要配置一些API的使能/禁用。

图7.19　Gpt General配置

①GptDeinitApi。

②GptEnableDisableNotificationApi。

③GptTimeRemainingApi。

④GptWakeupFunctionalityApi等。

（2）GptChannelConfigSet配置

切换到GptChannelConfigSet界面，点击“+”Add new element with default values可添加GptChannelConfigSet，如图7.20（左）所示。双击GptChannelConfigSet_0的Index可进入如图7.20（右）所示的GptChannelConfiguration配置界面。

图7.20　GptChannelConfigSet配置

如前所述，本书示例使用GPT通道为OS提供计时功能，所以添加一个GptChannel，如图7.21所示为该通道的配置界面。

图7.21　GptChannelConfigSet→GptChannelConfiguration配置

其中，各配置项意义描述如下。

①GptChannelId：Gpt通道Id号。

②GptHwChannel：Gpt硬件通道，其中，基于3个eTimer（Enhanced Motor Control Timer）模块有18个通道，基于STM（System Timer Module）有4个通道，基于PIT（Periodic Interrupt Timer）有4个通道，本书示例使用STM_0_CH_0。

③GptChannelMode：Gpt通道模式，如前所述分为连续模式（CONTINUOUS）和单次模式（ONESHOT），这里使用连续模式。

④GptChannelTickFrequency：Gpt通道频率，其中以Tick为计数单位。

⑤GptChannelClkSrcRef：Gpt通道时钟源参考，需要引用GptClockReferencePonit。

⑥GptStmPrescaler（_Alternate）：基于STM的分频。

⑦GptEtimerChannelClkSrc（_Alternate）：基于eTimer的分频。

⑧GptChannelTickValueMax：最大计数值。

⑨GptFreezeEnable：硬件资源冻结使能。

⑩GptEnableWakeup：Gpt通道唤醒使能。

⑪GptNotification：Gpt通知函数，即定时器到达设定值将调用到该函数。

（3）GptClockReferencePonit配置

GptClockReferencePonit界面中，可以点击“+”Add new element with default values添加GptClockReferencePonit，

这里需要引用先前在Mcu模块中定义的McuClockReferencePoint，如图7.22所示。

图7.22　GptClockReferencePonit配置

（4）Gpt模块常用接口函数介绍

Gpt模块在使用过程中需要初始化、使能Gpt通知函数，并且开启Gpt通道。Gpt通知函数名虽然在上述配置中已经配置了，但是需要自己进行函数的实现。

①函数Gpt_Init：

void Gpt_Init（const Gpt_ConfigType *configPtr）；

参数：const Gpt_ConfigType *。

返回值：void。

②函数Gpt_EnableNotification：

void Gpt_EnableNotification（Gpt_ChannelType channel）；

参数：Gpt_ChannelType。

返回值：void。

③函数Gpt_StartTimer：

void Gpt_StartTimer（Gpt_ChannelType channel， Gpt_ValueType val

参数：Gpt_ChannelType、Gpt_ValueType，后者需要填入计数值，即GptChannelTickValue。

返回值：void。

本书示例中Gpt通道使用连续模式，所以Gpt通道只需要初始化一次即可，当计数值到达设定值时将自动清零，重新开始计数。这里将其作为操作系统的时钟源，即产生OS的Tick。其中，GptNotification配置项所定义的函数名为Gpt_Cbk_ProcessOsCounter，具体实现如下，函数中调用操作系统接口函数IncrementCounter（Rte_TickCounter）产生Tick。

```
    Gpt_Init（&GptChannelConfigSet_0）；

Gpt_EnableNotification（GptConf_GptChannelConfiguration
_OsCounter）；
Gpt_StartTimer（GptConf_GptChannelConfiguration_OsCount
er，OSCYCLESPERSECOND）；

void Gpt_Cbk_ProcessOsCounter（void）
{
    IncrementCounter（Rte_TickCounter）；
}
```

7.2.3 Port模块

PORT驱动（PORT Driver）主要是对微控制器的PORT模块进行初始化配置。由于单片机的引脚存在复用，如MPC5744P单片机A［0］引脚功能如图7.23所示，它可用作通用输入/输出口（General Purpose Input Output，GPIO）、串行外设接口（Serial Peripheral Interface，SPI）的时钟引脚（Serial Clock，SCK）等。所以，需要根据具体应用对单片机各引脚属性进行相关配置。

Port Pin	SIUL2 MSCR/ IMCR Number	MSCR/ IMCR SSS Value[1]	Signal	Module	Short Signal Description	Dir	LQFP144	BGA257
A[0]	MSCR[0]	0000 (Default)[2]	GPIO[0]	SIUL2-GPIO [0]	General Purpose IO A[0]	I/O	73	P12
		0001	ETC0	e Timer_0	e Timer_0 Input/Output Data Channel 0	I/O		
		0010	SCK	DSPI2	DSPI2 Serial Clock (output)	I/O		
		0011-1111	—	Reserved	—	—		
	IMCR[48]	0001	SCK	DSPI2	DSPI2 Serial Clock (input)	I/O		
	IMCR[59]	0010	ETC0	e Timer_0	eTimer_0 Input Data Channel	I/O		
	IMCR[173]	0001	REQ0	SIUL2	SIUL2 External Interrupt 0			

图7.23　MPC5744P单片机A［0］引脚功能

Port模块主要就是对单片机各引脚属性的配置，可配置的参数包括引脚的方向（输入或输出）、运行期间引脚方向的可变性、引脚的工作模式、运行期间引脚工作模式的可变性、引脚的初始值、内部上拉的激活等。该模块配置需要结合硬件原理图上对于单片机各引脚的功能需求定义，并参考MPC5744P单片机手册中的Pin muxing表格。

（1）Port General配置

Port　General配置主要是对Port模块整体功能的配置，如图7.24所示。其中，主要配置项如下。

图7.24　PortGeneral配置

①Port Development Error Detect：Port模块开发错误检测使能。

②Port SetPinDirection Api：Pin方向设置API使能。

③Port SetPinMode Api：Pin模式设置API使能等。

（2）PortConfigSet配置

PortConfigSet菜单主要是对单片机引脚属性的定义，可以点击“+”Add new element with default values添加Port配置，如图7.25所示。

图7.25　PortConfigSet配置

双击PortConfigSet_0的Index可进入如图7.26所示的PortConfigSet界面，其中有两项内容。

图7.26　PortConfigSet→General配置

①PortConfigSet→General：可针对未使用的Pin的模式进行全局配置，如图7.26所示。

②PortConfigSet→PortContainer：可配置所使用的Pin的一些属性。

当切换到PortContainer界面后，可以点击“+”Add new element with default values添加Port，如图7.27（左）所示，这里的Port和芯片引脚没有明确的对应关系。双击一个Port的Index可为其添加Pin，如图7.27（右）所示。

图7.27　PortConfigSet→PortContainer配置

双击PortPin的Index可进入PortPin属性配置界面，如图7.28所示。主要配置项及其说明如下。

图7.28　PortConfigSet→PortContainer→PortPin配置

①PortPin Wpe：Enable Weak Pull Up/Down，内部上拉/下拉使能。

②PortPin Wps：在勾选PortPin Wpe的情况下，勾选PortPinWps代表开启内部上拉，未勾选PorPinWps代表内部下拉。

③PortPin Ode：开漏（Open-Drain）输出使能。

④PortPin Safe Mode：安全模式使能。

⑤PortPin With Read Back：回读功能使能。

⑥PortPin Hysteresis Control：输入迟滞使能。

⑦PortPin Direction Changeable：引脚方向可变使能。

⑧PortPin Mode Changeable：引脚模式可变使能。

⑨PortPin Id：引脚的Id号。

⑩PortPinPcr：描述引脚的PCR（Port Configuration Register）。

⑪PortPin Direction：定义引脚方向，输入（PORT_PIN_IN）、输出（PORT_PIN_OUT）、输入输出（PORT_PIN_INOUT）。

⑫PortPin Initial Mode：定义引脚初始模式，默认为GPIO。

⑬PortPin Mode：定义引脚模式，如图7.29所示为A［0］引脚模式可选项。

图7.29 A［0］引脚模式可选项

⑭PortPin Level Value：定义引脚初始化电平。

⑮PortPin Output Slew Rate：定义引脚电压转换速率。

（3）Port模块初始化函数介绍

当Port模块配置完成后，在实际使用中，需要对Port模块进行初始化。在AUTOSAR规范中，其初始化函数如下。

函数Port_Init：

void Port_Init（const Port_ConfigType *ConfigPtr）；

参数：const Port_ConfigType *。

返回值：void。

本书示例中Port模块初始化的代码实现如下。

```
Port_Init（&PortConfigSet_0）；
```

7.2.4 Dio模块

DIO驱动（Digital Input/Output Driver）对微控制器硬件引脚的访问进行了抽象，并且还可以对引脚进行分组。DIO驱动对于微控制器引脚的数据读写操作都是同步的。

在AUTOSAR中，将一个单片机数字I/O引脚（Pin）定义为DIO通道（DIO Channel）；可把若干个DIO通道通过硬件分组成为一个DIO端口（DIO Port）。DIO端口中相邻几个DIO通道的逻辑组合则称为DIO通道组（DIO Channel Group），在配置过程中可以设置寄存器位屏蔽值、位偏移量等，从而对多个数字I/O引脚同时进行读/写操作，如图7.30所示。

图7.30　Dio Channel Group

Dio模块中涉及的DIO Channel，即单片机引脚（Pin），若要正常使用，必须在Port模块中对该引脚进行属性配置，即配置为GPIO（General Purpose I/O）模式。

（1）Dio General配置

Dio General配置主要是对Dio模块整体功能的配置，如图7.31所示。其中，主要配置项如下。

图7.31　Dio General配置

①Dio Development Error Detect：Dio模块开发错误检测使能。

②Dio Read Zero For Undefined Port Pins：读取未定义引脚值为0等。

（2）DioConfig配置

切换至DioConfig界面，可以点击“+”Add new element with default values添加Dio配置，如图7.32（左）所示。双击DioConfig_0的Index，可进入如图7.32（右）所示的DioPort配置界面，点击“+”可以新建DioPort。

图7.32　DioConfig配置

双击DioPort的Index，可见每个DioPort配置又有三项内容。

①DioPort→General：为当前DioPort设定一个Dio　Port　Id。工具规定，对于单片机PortA=0、PortB=1…PortI=8、PortJ=9，如图7.33（左）所示。

②DioPort→DioChannel：点击“+”可以添加DioChannel，即DIO通道；这里的Dio Channel Id是基于Dio Port Id而言的，如Dio Port Id为0、Dio Channel Id为0则表示引脚PA［0］，以此类推。如图7.33（右）所示为A型车灯的配置，其通过数字输出DO直接控制车灯的亮灭。

图7.33　DioConfig→DioPort配置

③DioPort→DioChannelGroup：为前述DIO通道组的配置。

（3）Dio模块常用接口函数介绍

在AUTOSAR规范中，Dio模块的常用接口函数有Dio_WriteChannel（写DIO通道状态）、Dio_ReadChannel（读DIO通道状态）、Dio_FlipChannel（变换DIO通道状态）等。下面结合A型车灯初始化与IOAbstractionSWC中与Dio模块相关的具体实例进行讲解。

①函数Dio_Init：

void Dio_Init（const Dio_ConfigType *ConfigPtr）；

参数：const Dio_ConfigType *，由于Dio模块只有Per Compile模式，所以其传入参数必须为NULL_PTR。

返回值：void。

```
Dio_Init（NULL_PTR）；
```

②函数Dio_WriteChannel：

void Dio_WriteChannel（const Dio_ChannelType ChannelId，const Dio_LevelType Level）；

参数：const Dio_ChannelType，即传入DIO通道Id号；const Dio_LevelType，即传入DIO通道状态值，STD_HIGH为1；STD_LOW为0。

返回值：void。

```
FUNC（void，IOAbstractionSWC_CODE）RE_SetLightState
（
  VAR（UInt8，AUTOMATIC）DESetLightState
）
```

```
{
    /* ----------------------------------------Server Call Point ---------------------------------- */
    Dio_WriteChannel（DioConf_DioChannel_Do_FrontLight, DESetLightState）;
}
```

这里需指出，配置完DioChannel后，在生成MCAL配置代码过程中会在Dio_Cfg.h文件中通过宏定义（#define）的方式将DioChannel Name与DioChannelId关联起来。例如：

#define DioConf_DioChannel_Do_FrontLight（（uint8）0x03U）

③函数Dio_ReadChannel：

Dio_LevelType Dio_ReadChannel（const Dio_ChannelType ChannelId

参数：const Dio_ChannelType，即传入DIO通道Id号。

返回值：Dio_LevelType，即返回当前通道状态，STD_HIGH为1；STD_LOW为0。

```
    FUNC（void，IOAbstractionSWC_CODE）RE_GetButtonState
（
        CONSTP2VAR（UInt8，AUTOMATIC，RTE_APPL_DATA）DEGetButtonState
）
{
    /* ----------------------------------------Server Call Point ---------------------------------- */
    *DEGetButtonState=
```

```
    Dio_ReadChannel（（Dio_ChannelType）DioConf_DioChannel_Din_FrontLightSwitch）；
}
```

④函数Dio_FlipChannel：

Dio_LevelType Dio_FlipChannel（const Dio_ChannelType ChannelId）

参数：const Dio_ChannelType，即传入DIO通道Id号。

返回值：Dio_LevelType，即返回当前通道状态，STD_HIGH为1；STD_LOW为0。

```
    FUNC（void，IOAbstractionSWC_CODE）RE_EcuAliveIndicator
（
        CONSTP2VAR（uint32，AUTOMATIC，RTE_APPL_DATA）channel,
        CONSTP2VAR（boolean，AUTOMATIC，RTE_APPL_DATA）state
）
{
    /*  ------------------------------------- Server Call Point  ---------------------------------  */
    *state=Dio_FlipChannel（（Dio_ChannelType）（*channel））；
}
```

7.2.5 Adc模块

在工程实践中，一般会将各类信号经过一些电路转换为电压类型的模拟信号输入到微控制器的模/数转换单元（Analog-to-Digital Converter，ADC），将模拟信号转换为数字信号后，可在单片机内部进行运算处理。ADC驱动（Analog-to-Digital Converter Driver）对微控制器内部模/数转换单元进行初始化和转换控制。它可以提供启动和停止模/数转换的服务，分别用来开启和禁用模/数转换的触发源。

ADC驱动在ADC通道（ADC Channel）的基础上进行。ADC通道把模拟信号输入引脚、所需的ADC电路和转换结果寄存器三部分联系成为一个整体，使其能被ADC驱动所控制与访问。此外，属于同一个ADC硬件单元（ADC HW Unit）的一个或者多个ADC通道，可以组成一个ADC通道组（ADC Channel Group），由同一触发源触发。但一个ADC通道组必须至少包含一个ADC通道。

ADC模块支持以下两种转换模式。

①单次转换（One-Shot Conversion）：ADC通道组中每个ADC通道只执行一次转换。

②连续转换（Continuous Conversion）：在启动转换后，ADC通道组将会自动重复进行转换，而不需要再次触发。

ADC模块可以选择以下两种触发源。

①软件触发（SW-TRIGGER）：ADC通道组通过ADC模块提供的服务来启动/停止转换，其可在上述两种转换模式下使用。

②硬件触发（HW-TRIGGER）：ADC通道组通过硬件事件（如边沿触发、定时器等）来启动转换，但该方式只能用于单次转换模式。

（1）Adc General配置

Adc General配置主要是对Adc模块整体功能的配置，如图7.34所示。其中，主要配置项如下。

图7.34　Adc General配置

①Adc Development Error Detection：Adc模块开发错误检测使能。

②Adc_ReadGroup API：读取ADC通道组API使能。

③Adc_StartStopGroup API：ADC通道组转换启动/停止API使能。

④Adc Priority Mechanism：ADC优先级机制选择。

⑤Adc Result Alignment：ADC转换原始结果对齐方式选择。

⑥Adc Timeout：ADC转换超时时间。

⑦Adc Max Queue Depth：ADC转换请求队列最大长度等。

（2）AdcConfigSet配置

切换至AdcConfigSet界面，可以点击“+”Add new element with default values添加Adc配置，如图7.35（左）所示。双击AdcConfigSet_ADC0的Index，可进入如图7.35（右）所示AdcHwUnit配置界面，点击“+”可以新建AdcHwUnit配置。MPC5744P单片机有4个ADC硬件单元：ADC_0、ADC_1、ADC_2、ADC_3。本书A型车灯示例用到了其中一个硬件单元ADC_0，所以这里新建一个AdcHwUnit即可。

图7.35　AdcConfigSet配置

双击AdcHwUnit_0的Index，可以对ADC硬件单元进行相关配置。

在AdcHwUnit→General界面（图7.36），配置重点是Adc Transfer Type和Adc Hardware Unit。

图7.36　AdcConfigSet→AdcHwUnit→General配置

①Adc Transfer Type：ADC转换类型，可采用中断方式ADC_INTERRUPT与DMA模式，这里采用ADC_INTERRUPT方式。

②Adc Hardware Unit：选择ADC硬件单元，此处为ADC_0。

切换到AdcHwUnit→AdcChannel界面，可以点击“+”Add new element with default values添加属于硬件单元ADC_0的ADC通道，如图7.37所示。

图7.37　AdcConfigSet→AdcHwUnit→AdcChannel配置

对于一个ADC通道而言，需要设定Adc Channel Resolution，即ADC通道的精度，这里使用12位精度，如图7.38所示。

图7.38　AdcConfigSet→AdcHwUnit→AdcChannel→General配置

如前所述，属于同一个ADC硬件单元的一个或者多个ADC通道可以组成一个ADC通道组，由同一触发源触发。在AdcHwUnit→AdcGroup界面，就需要完成ADC通道组的配置及其中所包含的ADC通道的添加。可以点击“+”Add new element with default values添加AdcGroup，如图7.39所示。

图7.39　AdcConfigSet→AdcHwUnit→AdcGroup配置

本书示例使用一个ADC通道组，其中包含前面所定义的所有ADC通道。对于这个ADC通道组的配置如图7.40所示。

图7.40　AdcConfigSet→AdcHwUnit→AdcGroup→General配置

在AdcGroup→General界面中，需要配置ADC通道组的属性，主要有以下几种。

①Adc Group Access Mode：ADC转换结果寄存器访问模式，有SINGLE-ACCESS和STREAMING-ACCESS模式，本书中示例采用SINGLE-ACCESS。

②Adc Group Conversion Mode：ADC通道组转换模式，有单次转换（ONESHOT）与连续转换（CONTINUOUS）模式，本书中示例采用单次转换模式。

③Adc Group Conversion Type：ADC通道组转换类型，有NORMAL与INJECTED模式，本书中示例采用NORMAL模式。

④Adc Group Id：ADC通道组Id号；

⑤Adc Group Trigger Source：ADC通道组触发源，有硬件触发与软件触发两种，本书中示例采用软件触发模式。

切换到AdcGroup→AdcGroupDefinition界面，可以点击“+”Add new element with default values添加上述ADC通道组中的ADC通道，如图7.41所示。

图7.41　AdcConfigSet→AdcHwUnit→AdcGroup→AdcGroupDefinition配置

（3）Adc模块常用接口函数介绍

在使用Adc模块之前，需要先进行模块的初始化，并建立转换结果缓存。

①函数Adc_Init：

void Adc_Init（const Adc_ConfigType*pConfigPtr）；

参数：const Adc_ConfigType*。

返回值：void。

②函数Adc_SetupResultBuffer：

Std_ReturnType　Adc_SetupResultBuffer（Adc_GroupType　Group，Adc_ValueGroupType

*pDataBufferPtr）；

参数：Adc_GroupType和Adc_ValueGroupType*。

返回值：Std_ReturnType。

Adc模块有两种转换结果的访问模式，由以下两个接口函数实现。

①函数Adc_ReadGroup：

Std_ReturnType Adc_ReadGroup（Adc_GroupType Group，Adc_ValueGroupType

*pDataBufferPtr）；

参数：Adc_GroupType与Adc_ValueGroupType*。

返回值：Std_ReturnType。

②函数Adc_GetStreamLastPointer：

Adc_StreamNumSampleType Adc_GetStreamLastPointer（Adc_Group

Adc_ValueGroupType**PtrToSamplePtr）；

参数：Adc_GroupType与Adc_ValueGroupType**。

返回值：Adc_StreamNumSampleType。

其次，还有一些ADC通道组启动、停止、转换状态回读相关的接口函数。

①函数Adc_StartGroupConversion：

void Adc_StartGroupConversion（Adc_GroupType Group）；

参数：Adc_GroupType。

返回值：void。

②函数Adc_StopGroupConversion：

void Adc_StopGroupConversion（Adc_GroupType Group）；

参数：Adc_GroupType。

返回值：void。

③函数Adc_GetGroupStatus：

Adc_StatusType Adc_GetGroupStatus（Adc_GroupType Group）；

参数：Adc_GroupType。

返回值：Adc_StatusType，有4种状态，即ADC_IDLE、ADC_BUSY、ADC_COMPLETED、ADC_STREAM_COMPLETED。

本书中，A型车灯控制器ADC相关代码实现如下。

```
VAR（Adc_ValueGroupType，AUTOMATIC）AdcRawBuffer［6］；
Std_ReturnType retValue=RTE_E_OK;

retValue=Adc_SetupResultBuffer（AdcConf_AdcGroup_AdcGroup_0, AdcRawBuffer）；
Adc_StartGroupConversion（AdcConf_AdcGroup_AdcGroup_0）；

FUNC（void，IOAbstractionSWC_CODE）RE_GetLightState
（
     CONSTP2VAR（UInt8，AUTOMATIC，RTE_APPL_DATA）DEGetLightState
）
{

  if（ADC_STREAM_COMPLETED==Adc_GetGroupStatus（AdcConf_AdcGroup_AdcGroup_0））
```

```
  {
    Adc_StopGroupConversion（AdcConf_AdcGroup_AdcGro
up_0）；
    if（ADC_IDLE==Adc_GetGroupStatus（AdcConf_AdcGrou
p_AdcGroup_0））
    {
        *DEGetLightState=（uint8）（AdcRawBuffer［ADC
0_AN6］>>4）；
    }
    else{…}
  }
  else{…}
}
```

7.2.6 Pwm模块

PWM驱动（Pulse Width Modulation Driver）为微控制器中的PWM模块提供初始化和控制服务，可产生占空比和周期都可改变的脉冲。

（1）Pwm General配置

Pwm General配置主要是对Pwm模块整体功能的配置，如图7.42所示。其中，主要配置项如下。

图7.42　Pwm General配置

①PwmDeInitApi：Pwm模块反初始化API使能。

②PwmSetDutyCycle：设置PWM波占空比API使能。

③PwmSetPeriodAndDuty：设置PWM波周期和占空比API使能。

④PwmDevErorDetect：Pwm模块开发错误检测使能。

⑤PwmNotificationSupported：Pwm通知函数使能。

⑥PwmClockRef：Pwm模块参考时钟引用，需引用先前在Mcu模块中定义的McuClockReferencePoint_MC_CLK。

（2）PwmChannelConfigSet配置

切换至PwmChannelConfigSet界面，可以点击“+”Add new element with default values添加PwmChannelConfigSet配置，如图7.43（左）所示。双击PwmChannelConfigSet_0的Index，可进入如图7.43（右）所示的PwmChannel配置界面，点击“+”可以新建PwmChannel配置。本书中B型车灯示例采用PWM输出来控制灯的亮度，所以需要配置一个PWM通道Pwm_Chn0_FrontLight。

图7.43　PwmChannelConfigSet配置

对于PwmChannel需要定义PWM通道的一些属性，如图7.44所示。

图7.44　PwmChannelConfigSet→PwmChannel配置

其中各配置说明如下。

①PwmChannelId：PWM通道Id号。

②Pwm Hw IP：PWM硬件单元选择，可选择Enhanced Motor Control Timer（eTimer）和Motor Control Pulse Width Modulator Module（FlexPWM）。

③PwmFlexPwmChannel：若选择FlexPwm Hardware IP，需要选择一个FlexPwmChannel配置。

④PwmeTimerChannel：若选择eTimer Hardware IP，需要选择一个eTimerChannel配置。

⑤Default Period：PWM波默认输出周期。

⑥PwmChannelClass：PWM通道类别，分为FIXED_PERIOD、FIXED_PERIOD_SHIFTED、VARIABLE_PERIOD。

⑦PwmPolarity：PWM波极性。

⑧PwmDutycycleDefault：PWM波默认占空比。

⑨PwmIdleState：PWM通道Idle状态的电平，PWM_HIGH或PWM_LOW。

⑩PwmNotification：PWM通道通知函数名。

MPC5744P单片机有两个FlexPwm模块：FlexPwm_0和FlexPwm_1。每个FlexPwm模块又具有四个Sub-Module：Sub-Module_0、Sub-Module_1、Sub-Module_2、Sub-Module_3。每个Sub-Module可输出三路PWM：PWM_A、PWM_B、PWM_X。MPC5744P单片机FlexPwm模块示意如图7.45所示。

图7.45　MPC5744P单片机FlexPwm模块示意

本书B型车灯示例采用FlexPwm_0、Sub-Module_0、PWM_A通道输出PWM，所以需要进行相关配置，如图7.46～图7.48所示。

图7.46　PwmChannelConfigSet→PwmFlexPwm→FlexPwm Modules配置

图7.47
PwmChannelConfigSet→PwmFlexPwm→PwmFlexPwmSubModules→FlexPwm SubModules配置

图7.48

PwmChannelConfigSet→PwmFlexPwm→PwmFlexPwmChannels→PwmFlexPwmChannels配置

对于Sub-Module，需要进行一些属性配置，如图7.49所示。

图7.49　PwmChannelConfigSet→PwmFlexPwmSubModulesSettings配置

（3）Pwm模块常用接口函数介绍

在使用Pwm模块之前，需要进行Pwm模块初始化。

①函数Pwm_Init：

void Pwm_Init（const Pwm_ConfigType*ConfigPtr）；

参数：const Pwm_ConfigType*。

返回值：void。

②函数Pwm_SetDutyCycle：

void　Pwm_SetDutyCycle（Pwm_ChannelType　ChannelNumber，uint16 DutyCycle）；

参数：Pwm_ChannelType和uint16，前者为PWM通道，后者为占空比设定值，AUTOSAR规范中规定0x0000对应0%，0x8000对应100%。

返回值：void。

本书B型车灯示例中与PWM相关的代码实现如下。

```
uint 16 FrontLight_DutyCycleSetValue=0;
FUNC（void，  IOAbstractionSWC_CODE）RE_SetLightState
（
      VAR（UInt8，AUTOMATIC）DESetLightState
）
{
  /*  ------------------Server Call Point  ------------
------  */
  FrontLight_DutyCycleSetValue=0x8000*DESetLightStat
```

```
e/100;
  Pwm_SetDutyCycle（Pwm_Chn0_FrontLight, （uint16）
 （FrontLight_DutyCycleSetValue））；
}
```

7.2.7 Icu模块

ICU驱动（Input Capture Unit Driver）可以控制微控制器的输入捕获单元ICU，MPC5744P单片机的Icu模块可以提供如下服务：

①高/低电平时间测量（High Time/Low Time Measurement）；

②占空比测量（Duty Cycle Measurement）；

③周期性信号时间测量（Period Time Measurement）；

④信号边沿检测和通知（Edge Detection and Notification）；

⑤边沿计数（Edge Counting）；

⑥边沿时间戳捕获（Edge Time Stamping）；

⑦中断唤醒（Wake-up Interrupt）。

（1）Icu General配置

Icu General配置主要是对Icu模块整体功能的配置，如图7.50所示。

图7.50　Icu General配置

（2）IcuConfigSet配置

切换至IcuConfigSet界面，可以点击“+”Add new element with default values添加IcuConfigSet配置，如图7.51（左）所示。双击IcuConfigSet_0的Index，可进入如图7.51（右）所示的界面。

图7.51　IcuConfigSet配置

切换到IcuChannel，点击“+”可以新建一个IcuChannel。本书中B型

车灯示例采用ICU来测量车灯控制信号的占空比，所以需要配置一个ICU通道Pwm0A2_LightMeasure。进入IcuChannel配置后，主要需要完成如下配置，如图7.52所示。

图7.52　IcuConfigSet→IcuChannel配置

①IcuHwIP：选择ICU硬件通道，ETIMER或者SIUL2。

②IcueTimerChannelRef：若IcuHwIP选择ETIMER，需要引用ETIMER通道配置。

③IcuDefaultStartEdge：通道默认激活边沿。

④IcuMeasurementMode：选择ICU工作模式，本书示例使用SIGNAL_MEASUREMENT模式。

⑤IcuSignalMeasurementProperty：若使用

SIGNAL_MEASUREMENT模式，需要选择测量，本书示例测量DUTY_CYCLE（占空比）。

由于本书示例Icu模块IcuHwIP选择了ETIMER，所以需要配置eTimer通道。切换到IcueTimer界面，点击“+”可新建一个IcueTimer配置，如图7.53（左）所示。双击其Index可进入如图7.53（右）所示的界面，选择一个eTimer Hardware Module（eTimer硬件模块），根据所选的PortPin，此处选择单片机的EETIMER_2模块。

图7.53　IcuConfigSet→IcueTimer配置

选择了eTimer硬件模块后，需要配置它的属性，如图7.54所示。

图7.54　IcuConfigSet→IcueTimer→eTimerChannels配置

（3）IcuConfigSet配置

在使用Icu模块之前，需要对其进行初始化。

①函数Icu_Init：

void Icu_Init（const Icu_ConfigType*ConfigPtr）；

参数：const Icu_ConfigType*。

返回值：void。

②函数Icu_GetDutyCycleValues：

void Icu_GetDutyCycleValues（Icu_ChannelType Channel，Icu_DutyCycleType*DutyCycleValues）；

参数：Icu_ChannelType和Icu_DutyCycleType*，前者为ICU通道，后者为采样结果缓存结构体，其定义如下。

```
typedef struct
{
    Icu_ValueType ActiveTime;    /* <@brief Low or High time value.*/
    Icu_ValueType PeriodTime;    /* <@brief Period time value.*/
}Icu_DutyCycleType;
```

返回值：void。

本书B型车灯示例中与ICU相关的代码实现如下。

```
UInt8 FrontLight_DutyCycleGetValue=0;

FUNC（void，IOAbstractionSWC_CODE）RE_GetLightState
（
     CONSTP2VAR（UInt8，AUTOMATIC，RTE_APPL_DATA）DEGet
LightState
）
{
/*Local Data Declaration*/
VAR（Icu_DutyCycleType, AUTOMATIC）Iuws_DutyCycle;

/*PROTECTED REGION ID（User Logic：RE_GetLightState）EN
ABLED START*/
/*Start of user code-Do not remove this comment*/

Icu_GetDutyCycleValues（PwmOA2_LightMeasure, &Iuws_
DutyCycle）;
FrontLight_DutyCycleGetValue=（Iuws_DutyCycle.Activ
eTime*100）/Iuws_DutyCycle.PeriodTime;
*DEGetLightState=FrontLight_DutyCycleGetValue;

/*End of user code-Do not remove this comment*/
/*PROTECTED REGION END*/
}
```

7.2.8 Can模块

CAN驱动（CAN Driver）属于微控制器抽象层中的通信驱动，它针

对的是微控制器内部的CAN控制器，可以实现对CAN控制器的初始化、发送/接收CAN报文、对接收报文的指示与对发送报文的确认、唤醒检测、溢出和错误处理等功能。CAN驱动可以访问硬件，并向上层提供独立于硬件的API。

（1）Can General配置

Can General配置主要是对Can模块整体功能的配置，如图7.55所示。其中，主要配置项如下。

图7.55　CanGeneral配置

①Can Change Baudrate Api：改变CAN波特率API使能。

②Development Error Detection：Can模块开发错误检测使能。

③Can Driver Index：CAN驱动Id号。

④Can Main Function Busoff Period：Can_Main Function_Bus OFF（）函数调用周期。

⑤Can MainFunction Wakeup Period：Can_MainFunction_Wakeup（）函数调用周期。

⑥Can Main Function Mode Period：Can_MainFunction_Mode（）函数调用周期。

⑦Can Multiplexed Transmission：双路复用功能的传输使能。

⑧Can Identical Id Cancellation：取消挂起的发送报文使能。

⑨Can Extended Id Support：支持扩展Id使能。

⑩Message buffer data size：CAN MailBox装载数据长度等。

（2）CanConfigSet配置

切换至CanConfigSet界面，可以点击“+”Add new element with default values添加Can配置，如图7.56（左）所示。双击CanConfigSet_0的Index，可进入如图7.56（右）所示的CanConfigSet配置界面。

图7.56　CanConfigSet配置

从CanConfigSet配置界面可见，其中有两项配置内容。

①CanController：CAN控制器属性配置。

②CanHardwareObject：CAN硬件对象，即CAN MailBox（MB）配置。

首先，配置CanConfigSet→CanController，点击“+”可以新建CanController配置，双击Index可进入CanController→General配置界面，如图7.57所示。

图7.57　CanConfigSet→CanContoller→General配置

CanConfigSet→CanContoller→General中配置项说明如下。

①Can Hardware Channel：CAN硬件通道，MPC5744P单片机一共有3个CAN控制器，即FlexCAN_A、FlexCAN_B、FlexCAN_C，本书示例使用FlexCAN_A。

②Can Controller Activation：CAN控制器使能。

③Can Controller Id：CAN控制器Id号。

④Can Rx Processing Type：接收数据的处理方式，轮询（POLLING）或中断（INTERRUPT）。

⑤Can Tx Processing Type：发送数据的处理方式，轮询（POLLING）或中断（INTERRUPT）。

⑥Can BusOff Processing Type：CAN BusOff事件处理方式，轮询（POLLING）或中断（INTERRUPT）。

⑦Can Wakeup Processing Type：CAN Wakeup事件处理方式，轮询（POLLING）或中断（INTERRUPT）。

⑧Can Controller Default Baudrate：CAN控制器默认的波特率配置。

⑨Can CPU Reference Clock：CAN模块引用的时钟，即Mcu模块中配置的McuClockReferencePoint_CAN_CLK。

⑩CanRxFifoWarningNotification：RxFifo Warning通知函数。

⑪CanRxFifoOverflowNotification：RxFifo Overflow通知函数。

⑫Can Error Notification Enable：Can Error通知函数等。

在完成了CanContoller的通用配置后，需要配置CanContoller的波特率（Baudrate），切换到CanControllerBaudrateConfig界面，点击“+”可以新建CanControllerBaudrateConfig配置，双击Index，可进入CanControllerBaudrateConfig General配置界面，如图7.58所示。

图7.58　CanConfigSet→CanContoller→CanControllerBaudrateConfig配置

其中，主要配置项说明如下。

①Can Time Segments Checking：CAN时间段检测使能。

②Can Automatic Time Segments Calculation：自动时间段计算使能，若使能，则CanControllerPropSeg、CanControllerSeg1、CanControllerSeg2、CanControllerSyncJumpWidth将禁用。

③Can Controller Prescaller：CAN控制器时钟分频。

④Can Controller BaudRate Config Id：CAN控制器波特率配置Id号，被SetBaudrate API使用。

⑤Can Controller BaudRate：设置CAN控制器的波特率，本文示例使用100（Kbps）。

⑥Can Synchronization Segment：同步段的时间。

⑦Can Propagation Segment：传播段的时间。

⑧Can Phase Segment 1：采样点前的时间段。

⑨Can Phase Segment 2：采样点后的时间段。

⑩Can Resynch Jump Width：同步跳跃宽度（Synchronization Jump Width），用于重同步的时间。

对于CanContoller，还需要配置CanFilterMask，即滤波器掩码，如图7.59所示。但Filter只用于Rx Basic CAN类型的MB中。

图7.59　CanConfigSet→CanContoller→CanFilterMask配置

在完成CanContoller配置后，可进行CanHardwareObject的配置。如前所述，CAN硬件对象CanHardwareObject为CAN MailBox（MB）的抽象。切换到CanHardwareObject界面，可以点击“+”Add new element with default values添加CanHardwareObject。本书示例一共涉及两帧报文，一帧发送、一帧接收。这里需要创建两个MB来完成CAN报文的收发，如图7.60所示。

图7.60　CanConfigSet→CanHardwareObject配置（一）

如图7.61所示是接收类型的CanHardwareObject的配置，对于每个CanHardwareObject，主要需要配置如下内容。

图7.61　CanConfigSet→CanHardwareObject配置（二）

①Can Implementation Type：FULL CAN（一个MB只能发送或者接收一帧CAN报文）；BASIC CAN（一个MB可以发送或者接收多帧CAN报文）。

②Can Id Message Type：CAN Id的类型，标准帧

（Standard Identifier-11 bits）、扩展帧（Extended Identifier-29 bits）与混合模式（Mixed Mode）。

③CanIdValue（Message Id）：结合CanFilterMask，定义CAN报文接收Id范围。

④Can Object Id（MB Handle）：MB的Id号。

⑤Can MB Type：MB类型，接收（RECEIVE）或者发送（TRANSMIT）。

⑥Can Controller Reference：CAN控制器引用，本书示例涉及一个CAN控制器。

⑦Can Filter Mask Reference：引用滤波器掩码。

（3）CanMainFunctionRWPeriods配置

由于本书示例中接收/发送数据等的处理方式采用了轮询模式，即是通过周期性调用Can_MainFunction_Read（）和Can_MainFunction_Write（）函数来实现报文收发的，所以，CanMainFunctionRWPeriods配置就是来设定轮询周期的，此处都配置成0.001s，如图7.62所示。

图7.62 CanMainFunctionRWPeriods配置

（4）Can模块初始化函数介绍

和其他模块一样，Can模块需要初始化，其初始化函数如下。

函数Can_Init：

void Can_Init（const Can_ConfigType*Config）；

参数：const Can_ConfigType*。

返回值：void。

本书示例中Can模块初始化的代码实现如下。

```
Can_Init（&CanConfigSet_0）；
```

7.2.9 Base与Resource模块

Base与Resource这两个模块与具体功能无关，不需要额外配置。其中，Resoruce模块主要指示MCAL配置工具所支持的芯片信息，本书使用的开发板所用的芯片为MPC5744P_lqfq144。Base模块主要指示AUTOSAR规范版本以及MCAL工具版本信息，如图7.63所示。

图7.63　Base与Resource模块配置

7.3 MCAL配置验证与代码生成

在配置完所需MCAL模块之后，就可以进行配置验证与代码生成。MCAL配置工具的工具栏如图7.64所示。其中，右起第二个按钮为“Verify selected project”，点击之后将进行配置验证。右起第一个按钮为“Generate Code for the currently selected project”，点击之后将进行MCAL代码生成。

图7.64 MCAL配置工具的工具栏

MCAL工程最终生成的配置代码如图7.65所示。

名称	类型
Adc_Cfg.h	C/C++ Header F...
Adc_CfgDefines.h	C/C++ Header F...
Can_Cfg.h	C/C++ Header F...
CDD_Mcl_Cfg.h	C/C++ Header F...
Dio_Cfg.h	C/C++ Header F...
Gpt_Cfg.h	C/C++ Header F...
Icu_Cfg.h	C/C++ Header F...
Mcu_Cfg.h	C/C++ Header F...
modules.h	C/C++ Header F...
Port_Cfg.h	C/C++ Header F...
Pwm_Cfg.h	C/C++ Header F...

名称	类型
Adc_Cfg.c	C Source File
Adc_PBcfg.c	C Source File
Can_Cfg.c	C Source File
Can_PBcfg.c	C Source File
CDD_Mcl_Cfg.c	C Source File
CDD_Mcl_PBcfg.c	C Source File
Dio_Cfg.c	C Source File
Gpt_Cfg.c	C Source File
Gpt_PBcfg.c	C Source File
Icu_Cfg.c	C Source File
Icu_DmaNotification.c	C Source File
Icu_PBcfg.c	C Source File
Mcu_Cfg.c	C Source File
Mcu_PBcfg.c	C Source File
Port_Cfg.c	C Source File
Port_PBcfg.c	C Source File
Pwm_Cfg.c	C Source File
Pwm_PBcfg.c	C Source File

图7.65　MCAL工程最终生成的配置代码

7.4 本章小结

本章在介绍MPC5744P单片机微控制器抽象层MCAL配置工具安装方法及MCAL配置工程创建方法的基础上，对微控制器抽象层中各常用模块结合本书示例进行了较为详细的讲解。在各模块的介绍过程中，先介绍了各模块的作用，再进行基本配置方法的介绍，最后介绍了各模块常用的接口函数，并展示了本书示例中MCAL模块相关的代码实现。通过本章的学习，可以对MCAL各常用模块的作用、配置方法以及接口函数调用方法有一个较为全面的认识。

第8章　AUTOSAR工程代码集成与调试

在完成了AUTOSAR系统级、ECU级以及软件组件级相关开发与代码生成工作后，需要进行代码集成与调试。本章对AUTOSAR工程代码架构及其集成、编译链接和调试的方法进行扼要介绍，并展现一下本书示例的调试现象。

8.1　AUTOSAR工程代码架构与集成方法介绍

一套完整的符合AUTOSAR规范的ECU代码主要包括：

①应用层软件组件代码；

②运行时环境代码；

③基础软件代码。

基于本书示例开发所用的AUTOSAR系统解决方案，基础软件BSW由于开发工具原因，BSW除MCAL以外的代码由RTA-BSW和RTA-OS工具生成，而MCAL代码则由EB tresos Studio工具生成。除了RTA-OS工具直接调用编译器对OS代码进行编译之外，其他BSW代码均是源代码，包括源文件（.c）和头文件（.h）。需要注意的是，这些BSW代码由动态代码（Dynamic Code）和静态代码（Static Code）两部分组成，前者是由配置工具根据相关配置信息生成的代码，后者则是各BSW模块功能的具体实现代码。如图8.1所示为Com模块最终生成的代码。

图8.1　Com模块最终生成的代码

除了AUTOSAR工具链生成的代码以外，还需要添加一些附加的代码文件，如单片机启动代码等。最终，根据AUTOSAR代码架构可以得到一个AUTOSAR代码集成基本流程，如图8.2所示。

图8.2　AUTOSAR代码集成基本流程

8.2 代码编译链接

目前，针对PowerPC系列单片机的编译器主要有Wind River和Green Hills，本书示例采用Wind River编译器进行代码编译链接，并将其集成于Eclipse平台。A型车灯示例Wind River编译工程架构如图8.3所示。

图8.3 Wind River编译工程架构

右键点击工程名→Build Project即可进行工程架构，如图8.4所示。

图8.4　基于Eclipse的工程架构

8.3 代码调试

代码编译通过后，需要使用调试器将单片机可执行文件烧写到单片机内进行软件调试。本书示例基于MPC5744P开发板，使用Lauterbach调试器进行调试，并使用ETAS BUSMASTER工具进行CAN报文观测，调试设备实物如图8.5所示。

图8.5 调试设备实物

8.3.1 单片机可执行文件下载

打开Lauterbach TRACE32工具，点击File→Run Script后，选择mpc5744p.cmm脚本，如图8.6所示。之后将会弹出如图8.7所示的界面，点击Yes并进行单片机可执行文件选择，如图8.8所示。

图8.6　Lauterbach TRACE32导入脚本

图8.7　单片机可执行文件下载提示

图8.8　单片机可执行文件选择

单片机可执行文件下载完成后，可进入软件调试界面，点击运行“▲”即可开始在线调试，如图8.9所示。

图8.9　Lauterbach TRACE32调试界面

8.3.2　A型车灯调试现象

对于A型车灯而言，打开车灯开关即打开车灯，并需要将车灯型号（LightType）和车灯状态（LightState）信息反馈到CAN总线上。A型车灯被关闭时的调试现象如图8.10所示。

图8.10　A型车灯被关闭时的调试现象

当RE_LightRequest运行实体的接口函数Rte_Call_RPortGetButtonState_OPGetButtonState（&rtb_FunctionCaller）的形参rtb_FunctionCaller变为1时，则表明车灯开关处于开启状态，A型车灯控制器检测到车灯开关打开时的调试现象如图8.11所示。

图8.11　A型车灯控制器检测到车灯开关打开时的调试现象

此时，进入I/O抽象层的RE_SetLightState运行实体，会发现其形参DESetLightState变为1，通过调用Dio_WriteChannel（DioConf_DioChannel_Do_FrontLight，DESetLightState）函数可以设定DO通道的输出为高电平，即打开车灯，如图8.12所示。

图8.12　A型车灯控制器向DO通道输出高电平

最终，A型车灯打开时向CAN总线上发的报文情况如图8.13所示。

图8.13　A型车灯打开时向CAN总线上发的报文

8.3.3　B型车灯调试现象

对于B型车灯而言，需要结合车灯开关情况和外界环境光强信息来

控制车灯的亮度，并需要将车灯型号（LightType）和车灯状态（LightState）信息反馈到CAN总线上。

最终，打开B型车灯开关，外界环境“光线较弱”时的调试现象如图8.14所示；外界环境“光线中等”时的调试现象如图8.15所示；外界环境“光线较强”时的调试现象如图8.16所示。

图8.14　外界环境“光线较弱”时的调试现象

图8.15　外界环境“光线中等”时的调试现象

图8.16　外界环境“光线较强”时的调试现象

8.4 本章小结

本章首先介绍了符合AUTOSAR规范的车用控制器代码架构，并在此基础上介绍了代码集成的基本方法。之后，对本书示例所采用的Wind River编译器和Lauterbach调试器的使用方法进行了简要介绍。最后，展示了A型与B型车灯控制器软件的调试现象，验证了软件的正确性。通过本章的学习，可以对AUTOSAR代码架构有一个较为清晰的认识，并可以掌握基本的代码集成、编译链接与调试方法。

第9章　AUTOSAR与功能安全

作为当前汽车领域最流行的话题之一，AUTOSAR和功能安全（Functional　Safety）是国际和国内众多OEM及零部件供应商竞相研究的两大热点。这两个新领域引入了众多的新概念，AUTOSAR以其面面俱到的规范文件给软件工程师们带来了繁杂的工作，而功能安全也以庞杂的体系要求及含混的可操作性给身处其中的工程师们带来了巨大的挑战。清晰地理解两者之一就已经有点力不从心。对于未曾同时接触过AUTOSAR或功能安全相关话题的工程师而言，试图弄清楚两者之间的直接关系就更加困难了。

本章试图从宏观上阐述两者之间的关联，给软件工程师一个全局的概念。从而可以更好地理解两者之间的关系，而非关注各自的标准解读或者指导读者如何才能实现或达到各自的要求——因为要达到这个宏大的目标，需要专业的咨询服务，及时间、人力、资金等众多成本的投入。

9.1 AUTOSAR对ISO 26262中支持部分的要求概述

首先，读者需要知道这两个规范AUTOSAR R4.x和ISO 26262《道路车辆功能安全》（其概览如图9.1所示）各自的要求，主要内容及意图是什么。概括来说，AUTOSAR定义了系统尤其是软件架构规范，它注重模块化软件设计、软硬件独立开发（Modularity）、模块的可重用性（Reusability）、标准化接口、模块可替换性（Exchangeability）等，以及定义了软件实现的一些方法。AUTOSAR试图通过标准统一的接口成本最小化地无缝替换或适配各种硬件环境，并尽量在各个控制器中重用各种已开发的应用算法。而功能安全是一个体系，从文化、流程、系统、软硬件开发等多方面提出了诸多要求，期望让产品的功能更安全。简单来说，AUTOSAR是一个软件架构，而ISO 26262强调的是从产品定义→软硬件开发→测试→生产的一个安全开发流程和生产管理等体系。从这个角度看，功能安全规范定义的范围更“大”一些！事实上，两者之间并不是直接的并列或者包含的关系，而只是有部分关联，且两者有关联的交集部分可类比于需求和实现的关系，即ISO 26262在软件方面从流程到开发，从方法到工具提了很多要求，而AUTOSAR恰巧在某些方面（尤其是软件架构部分）实现或者满足了部分功能安全的要求。

图9.1　ISO 26262标准概览

虽然AUTOSAR R4.x对应用层做了部分标准化，但更多的规范是定义了底层软件的通用功能。下面将详细地列出功能安全在软件方面的部分要求，及对应的AUTOSAR R4.x规范相关的实现或满足的内容，以帮助读者对两者之间的关联部分有个初步的理解。

功能安全ISO 26262：2011按照通常的V开发模型顺序逐一阐述，其中对软件部分的要求主要集中在了Part 6部分。由于AUTOSAR偏重于软件架构，故与功能安全相关的多在于软件架构部分。另外，ISO 26262：2011硬件部分和系统测试部分多多少少也有一些要求与AUTOSAR的某些机制实现是相对应的，在此也一并阐述。

需要指出的是，尽管AUTOSAR提供了一些安全方法或机制来开发

安全相关的系统，但并不能保证依据AUTOSAR规范开发的软件就一定是满足功能安全目标的。因为要达到这个目标还需要其他方面的配合，只能说符合AUTOSAR规范的开发方法有助于最终产品更好地满足功能安全的要求。

以下内容将按照“功能安全要求→AUTOSAR对应实现”的形式来逐步讲述有关AUTOSAR R4.x对ISO 26262：2011能够提供支持的部分。

9.1.1 ISO 26262对架构设计的要求

ISO 26262中要求在软件架构设计时，软件架构需具备以下特征：

①可验证性；

②可追溯性；

③可配置性；

④灵活性；

⑤可测性；

⑥可维护性。

另外，为了避免由于较高的复杂度导致的软件失效（Failure），软件架构设计时还需考虑以下因素：

①模块化；

②封装化；

③简单化。

显然AUTOSAR出于其层次分明、模块独立性强以及其静态代码与配置代码相对独立等特点，几乎完全符合上述要求。因为AUTOSAR“天生”就具备以下几大特点：

①可重用性（Reusability），规定了模块间的标准接口；

②可验证性（Verifiability），依托于工具链提供商的检测；

③模块化（Modularity），软硬件的独立性及各层级模块间的独立性；

④封装性（Encapsulation），基于层级封装的设计。

ISO 26262对软件架构设计的静态设计部分，如架构框图、数据流、数据的类型及属性、模块接口和整个软件的接口信息等要求在AUTOSAR中由Compositions、SWCs、Interfaces、Ports、Runnables等要素来实现；对动态架构设计，如状态迁移图、时序流、Schedule图等要求则由对应的OS Tasks、RTE Events或SchM等要素来实现。

为了提升软件的鲁棒性，ISO 26262标准中要求软件架构设计时需要考虑错误检测机制和错误处理机制，如图9.2所示。

Table 4 — Mechanisms for error detection at the software architectural level

Methods		ASIL			
		A	B	C	D
1a	Range checks of input and output data	++	++	++	++
1b	Plausibility check[a]	+	+	+	++
1c	Detection of data errors[b]	+	+	+	+
1d	External monitoring facility[c]	o	+	+	++
1e	Control flow monitoring	o	+	++	++
1f	Diverse software design	o	o	+	++

[a] Plausibility checks can include using a reference model of the desired behaviour, assertion checks, or comparing signals from different sources.

[b] Types of methods that may be used to detect data errors include error detecting codes and multiple data storage.

[c] An external monitoring facility can be, for example, an ASIC or another software element performing a watchdog function.

Table 5 — Mechanisms for error handling at the software architectural level

Methods		ASIL			
		A	B	C	D
1a	Static recovery mechanism[a]	+	+	+	+
1b	Graceful degradation[b]	+	+	++	++
1c	Independent parallel redundancy[c]	o	o	+	++
1d	Correcting codes for data	+	+	+	+

[a] Static recovery mechanisms can include the use of recovery blocks, backward recovery, forward recovery and recovery through repetition.

[b] Graceful degradation at the software level refers to prioritizing functions to minimize the adverse effects of potential failures on functional safety.

[c] Independent parallel redundancy can be realized as dissimilar software in each parallel path.

图9.2　ISO 26262标准中对软件架构级别错误检测机制及错误处理机制的要求

相应地，AUTOSAR规范中提出了如下概念：

①存储空间隔离（Memory partitioning）；

②防御性行为（Defensive behavior）；

③端对端保护（End-to-end communication protection）；

④程序流监控（Program flow monitoring）；

⑤硬件测试和检查（Hardware testing and checking），其中硬件相关的检测在ISO 26262-5中有要求。

此外，功能安全中对软件架构设计的要求如图9.3所示。

Table 3 — Principles for software architectural design

Methods		ASIL			
		A	B	C	D
1a	Hierarchical structure of software components	++	++	++	++
1b	Restricted size of software components[a]	++	++	++	++
1c	Restricted size of interfaces[a]	+	+	+	+
1d	High cohesion within each software component[b]	+	++	++	++
1e	Restricted coupling between software components[a, b, c]	+	++	++	++
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts[a, d]	+	+	+	++

[a] In methods 1b, 1c, 1e and 1g "restricted" means to minimize in balance with other design considerations.

[b] Methods 1d and 1e can, for example, be achieved by separation of concerns which refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concept, goal, task, or purpose.

[c] Method 1e addresses the limitation of the external coupling of software components.

[d] Any interrupts used have to be priority-based.

图9.3　ISO 26262中对软件架构设计的要求

AUTOSAR对于软件架构要求的支持是显而易见的。AUTOSAR基于分层和模块化理念的设计，以及Interface、SWC等的灵活使用，使得软件架构满足1a层次化结构、1b限制软件组件数量、1c限制接口数量。另外，AUTOSAR出于模块化可剪裁等特点，模块间耦合度已经很低了，基本满足1e低耦合等要求。至于其1d高内聚，AUTOSAR引入Software Components，整个概念本身即建议一个SWC包含一个相关的功能块。1f合适的调度和1g限制中断使用是针对AUTOSAR的OS和RTE而言的，强大的OS功能和灵活的Event使用，可实现最合适的调度机制，以及用轮询的Event触发方式取代中断的使用。

需要注意的是，1a～1g部分要求在实际产品开发中牵涉到软件的具体设计和实现，且多为主观评价，在功能安全产品认证时需要结合各自公司和产品的衡量标准具体分析及评价。例如，在具体的项目实施中，应用层的ASW是否低耦合、高内聚主要取决于工程师的具体实现，而非一个空泛的软件组件定义。

9.1.2 ISO 26262对硬件验证的要求

ISO 26262在硬件验证方面也有一些要求，在硬件测试方面主要包括以下三个部分：

①RAM测试；

②Flash测试；

③内核测试。

在ISO 26262 Part 5中，定义了一系列RAM错误检测机制，如图9.4所示。

Table D.6 — Volatile memory

Safety mechanism/measure	See overview of techniques	Typical diagnostic coverage considered achievable	Notes
RAM pattern test	D.2.5.1	Medium	High coverage for stuck-at failures. No coverage for linked failures. Can be appropriate to run under interrupt protection
RAM March test	D.2.5.3	High	Depends on the write read order for linked cell coverage. Test generally not appropriate for run time
Parity bit	D.2.5.2	Low	—
Memory monitoring using error-detection-correction codes (EDC)	D.2.4.1	High	The effectiveness depends on the number of redundant bits. Can be used to correct errors
Block replication	D.2.4.4	High	Common failure modes can reduce diagnostic coverage
Running checksum/CRC	D.2.5.4	High	The effectiveness of the signature depends on the polynomial in relation to the block length of the information to be protected. Care needs to be taken so that values used to determine checksum are not changed during checksum calculation Probability is 1/maximum value of checksum if random pattern is returned

图9.4 ISO 26262中提出的RAM错误检测机制

这里RAM测试是为了测试RAM单元的物理健康性，不是为了测试RAM的内容。其中RAM单元是存储的单元，依赖于处理器。

对应地，AUTOSAR MCAL中有一个RamTst模块，专门实现了这一功能。其原理通常是在被测的RAM位置中写入一个已知的序列，然后读取它并测试读取序列是否与所写的序列相同。在执行RAM测试算法时，不允许其他软件修改测试的RAM区域。

ISO 26262 Part 5：D1部分提到Non-Volatile Memory，即ROM/Flash失效模式，其部分失效模式如图9.5所示。对应的FlsTst也是AUTOSAR架构中MCAL层的一个标准模块。它提供算法来测试非易失性存储区，其测试服务可以在上电初始化后的任一时候执行，并有用户根据不同的安全分析需求选择合适的测试算法及合适的执行点来满足系统的安全需求。

Non-volatile memory	D.5	Stuck-at[a] for data and addresses and control interface, lines and logic	d.c. fault model[b] for data and addresses (includes address lines within same block) and control interface, lines and logic	d.c. fault model[b] for data, addresses (includes address lines within same block) and control interface, lines and logic

图9.5　ROM/Flash失效模式（部分）

至于内核检测，ISO 26262 Part 5：D1部分提到处理器内核的失效模式，其部分失效模式如图9.6所示。

Processing units	ALU - Data Path	D.4/D.13	Stuck-at[a]	Stuck-at[a] at gate level	d.c. fault model[b] Soft error model[c] (for sequential parts)
	Registers (general purpose registers bank, DMA transfer registers…), internal RAM	D.4	Stuck-at[a]	Stuck-at[a] at gate level Soft error model[c]	d.c. fault model[b] including no, wrong or multiple addressing of registers Soft error model[c]
	Address calculation (Load/Store Unit, DMA addressing logic, memory and bus interfaces)	D.4/D.5/D.6	Stuck-at[a]	Stuck-at[a] at gate level Soft error model[c] (for sequential parts)	d.c. fault model[b] including no, wrong or multiple addressing Soft error model[c] (for sequential parts)
	Interrupt handling	D.4/D.10	Omission of or continuous interrupts	Omission of or continuous interrupts Incorrect interrupt executed	Omission of or continuous interrupts Incorrect interrupt executed Wrong priority Slow or interfered interrupt handling causing missed or delayed interrupts service
	Control logic (Sequencer, coding and execution logic including flag registers and stack control)	D.4/D.10	No code execution Execution too slow Stack overflow/underflow	Wrong coding or no execution Execution too slow Stack overflow/underflow	Wrong coding, wrong or no execution Execution out of order Execution too fast or too slow Stack overflow/underflow
	Configuration Registers	D.4	—	Stuck-at[a] wrong value	Corruption of registers (soft errors) Stuck-at[a] fault model
	Other sub-elements not belonging to previous classes	D.4/D.13	Stuck-at[a]	Stuck-at[a] at gate level	d.c. fault model[b] Soft error model[c] (for sequential part)

图9.6　处理器内核的失效模式（部分）

显然要求有相应的机制去检测有关失效模式。AUTOSAR架构中MCAL层标准模块CoreTst可通过对内核进行测试，从而能够验证CPU寄存器、中断控制器、算术逻辑单元、存储接口、缓存控制器、MPU（Memory Protection Unit）单元等的功能是否完整。

总之，AUTOSAR有专门的功能模块帮助用户实现功能安全对硬件的检测要求。当然，这些功能模块的实现强烈依赖于具体的硬件，且通常由MCAL的供应商负责提供。

9.1.3　ISO 26262对通信验证的要求

在ISO 26262 Part 5：D8中总结了对通信总线的要求（图9.7），如帧超时检测、节点丢失检测、CRC校验等，这些在AUTOSAR对应的总线模块如CAN协议栈、LIN协议栈等需求和实现中都有体现。

Table D.8 — Communication bus (serial, parallel)

Safety mechanism/measure	See overview of techniques	Typical diagnostic coverage considered achievable	Notes
One-bit hardware redundancy	D.2.7.1	Low	—
Multi-bit hardware redundancy	D.2.7.2	Medium	—
Read back of sent message	D.2.7.9	Medium	—
Complete hardware redundancy	D.2.7.3	High	Common failure modes can reduce diagnostic coverage
Inspection using test patterns	D.2.7.4	High	—
Transmission redundancy	D.2.7.5	Medium	Depends on type of redundancy. Effective only against transient faults
Information redundancy	D.2.7.6	Medium	Depends on type of redundancy
Frame counter	D.2.7.7	Medium	—
Timeout monitoring	D.2.7.8	Medium	—
Combination of information redundancy, frame counter and timeout monitoring	D.2.7.6, D.2.7.7 and D.2.7.8	High	For systems without hardware redundancy or test patterns, high coverage can be claimed for the combination of these safety mechanisms

图9.7　ISO 26262中对通信总线的要求

在对应的AUTOSAR ComSpec中，针对Timeout Monitor、Frame Counter等都有明确的要求和机制，散落在诸如［SRS_Com_02037］、［SRS_Com_00192］、［SWS_Com_00688］、［SWS_Com_00587］、［SWS_Com_00739］、［SWS_Com_00308］、［SWS_Com_00727］、［SWS_Com_00333］等。

数据序列控制器（Data Sequence Control）：AUTOSAR Com模块以I-PDU计数器的形式提供了数据序列控制机制。一旦侦测到计数器不连续，例如重复或丢失的情况，其I-PDU将被直接丢弃掉。

截止时间监视（Deadline Monitoring）：在信号组的截止时间监视

范围内，监控是否接收到信号。接收截止时间监视通过I-PDU组的控制，可以启用和禁用接收截止时间监视。在接收超时的情况下，Com模块可决定是否用初始值替换信号/信号组值，或保持最后一个接收值。例如，AUTOSAR标准中有如下要求。

①SRS_Com_02058：AUTOSAR Com模块应支持接收端对更新的信号/信号组的截止时间监控。

②SRS_Com_02099：AUTOSAR Com模块应提供一种检测出接收到的I-PDU失序的机制。

③SRS_Com_00192：AUTOSAR Com模块应支持启用和禁用I-PDU组的接收截止时间监控。

④SRS_Com_02037：AUTOSAR Com模块应对所有信号和信号组独立执行传输截止时间监控（如果配置）功能。

⑤SRS_Com_02030：AUTOSAR Com模块应支持检测收到的信号或信号组是否由发送方更新。

9.1.4 ISO 26262对FFI的要求

在ISO 26262 Part 6附录D中，从免于干扰（Freedom From Interference，FFI）出发，要求提供对应的机制来检测执行时间错误和数据交换错误，并在内存空间做到隔离保护等。

（1）D.2.2时序和执行

关于时间约束，在每个软件分区中执行的软件元素可以考虑以下故障的影响：

①执行阻塞；

②死锁（Deadlocks）；

③活锁（Livelocks）；

④错误的执行时间分配；

⑤软件要素之间同步不正确。

（2）D.2.3存储器（Memory）

对于存储器，在每个软件分区中执行的软件元素需考虑下面列出的错误的影响：

①内容损坏；

②对其他软件元素存储空间读写访问权限。

（3）D.2.4信息交互

关于信息交换，对每个发送方或接收方，需考虑下列故障的原因或错误的影响：

①重复接收；

②信息丢失；

③信息延迟；

④信息插入；

⑤信息伪装或信息的不正确寻址；

⑥信息序列错误；

⑦信息损坏；

⑧一个发送端给多个接收端的信息不对称；

⑨接收方仅接收到部分子集信息；

⑩通信信道访问阻塞。

在AUTOSAR架构中有不同的安全机制来检测和防范不同的错误，例如：

a.对存储器（Memory）的要求主要由存储器分区（Memory Partition）和存储器保护（Memory Protection）两种机制来实现；

b.对信息交互的保护主要由E2E（End-to-End Communication Protection）机制来实现；

c.对执行时间的保护要求一般是由WdgM（Watchdog Manager）的时间监控和操作系统OS的时间保护来满足的。

下文将会分小节来详细描述AUTOSAR中实现FFI的安全机制。

9.1.5 ISO 26262对编码风格的要求

ISO 26262中对编码风格的要求如图9.8所示。其中，主要包括：

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity[a]	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++
1c	Enforcement of strong typing[c]	++	++	++	++
1d	Use of defensive implementation techniques	o	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++

[a] An appropriate compromise of this topic with other methods in this part of ISO 26262 may be required.

[b] The objectives of method 1b are

— Exclusion of ambiguously defined language constructs which may be interpreted differently by different modellers, programmers, code generators or compilers.

— Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.

— Exclusion of language constructs which could result in unhandled run-time errors.

[c] The objective of method 1c is to impose principles of strong typing where these are not inherent in the language.

图9.8 ISO 26262中对编码风格的要求

①1a低复杂度的实施（Enforcement of low complexity）；

②1b语言子集（Use of language subsets）；

③1c强类型的实施（Enforcement of strong typing）；

④1d防御性实现技术的使用（Use of defensive implementation techniques）；

⑤1e已建立的设计原理的使用（Use of established design principles）；

⑥1f无歧义图示法的使用（Use of unambiguous graphical representation）；

⑦1g设计规范的使用（Use of style guides）；

⑧1h命名惯例（Use of naming conventions）等。

而上述许多编码风格的要求在AUTOSAR规范里几乎随处可见，如图9.9所示。

The following table provides a list of examples of ISO26262 Requirements mapped to the definition of AUTOSAR Basic Software.

ID	Functional Safety Measures	ISO Reference	AUTOSAR Requirement/Feature
001	Enforcement of strong typing	ISO26262-6 Table 1, 1c	AUTOSAR Meta-Model
002	Use of established design principles	ISO26262-6 Table 1, 1e	AUTOSAR Layered Architecture
003	Use of unambiguous graphical representation	ISO26262-6 Table 1, 1f	Standard representation of the AUTOSAR Meta-Model
004	Use of naming conventions	ISO26262-6 Table 1,1h	AUTOSAR Application Interfaces definition: AUTOSAR_MOD_AITable.xls AUTOSAR_EXP_AIUserGuide.pdf
005	Semi-formal Notation	ISO26262-6 Table 2, 1b	AUTOSAR Meta-Model
006	Restricted size of interfaces	ISO26262-6 Table 3, 1c	Per domain, application interfaces were proposed: AUTOSAR_EXP_AIBodyAndComfort.pdf AUTOSAR_EXP_AIChassis.pdf AUTOSAR_EXP_AIOccupantAndPedestrianSafety.pdf AUTOSAR_EXP_AIHMIMultimediaAndTelematics.pdf AUTOSAR_EXP_AIPowertrain.pdf
007	Restricted coupling between software components	ISO26262-6 Table 3, 1e	AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf Please see: Interfaces: General Rules Layer Interaction Matrix.
008	Restricted use of interrupts	ISO26262-6 Table 3, 1g	AUTOSAR_EXP_ InterruptHandlingExplanation.pdf
009	Detection of data errors	ISO26262-6 Table 4, 1c	AUTOSAR_SWS_E2ELibrary.pdf AUTOSAR_SWS_CRCLibrary.pdf
010	Control flow monitoring	ISO26262-6 Table 4, 1e	AUTOSAR_SWS_WatchdogManager.pdf

图9.9　AUTOSAR规范中对编码风格的要求

下面以1d（防御性实现技术的使用）为例进行介绍，在AUTOSAR规范中就有多处提及，如图9.10所示。

BSW modules shall tolerate concurrent access to HW registers using defensive behavior and the techniques like:
- Protecting the read-modify-write access from interruption
- Using atomic (non-interruptible) instructions for read-modify-write access
- Protecting the access to set of registers, which have to be modified together, from interruption](*SRS_BSW_00451*)

4.2.35 [SRS_Diag_04107] Defensive behavior of the DEM module

[

Type:	Valid
Description:	For safety-related applications, the Diagnostics Event Manager shall ensure data integrity of errors information stored in non-volatile memory.
Rationale:	Protection of error events memory is needed for safety-related
Use Case:	Error events memory could have been corrupted
Dependencies:	--
Supporting Material:	Use the optional CRC and redundancy capabilities provided by the NVRAM Manager for Diagnostics Event Manager NVRAM Blocks. Only blocks assigned to error events of high severity can be protected. These blocks can be stored in non-volatile memory when the error event is confirmed (before shutdown of the ECU), refer to RS_BRF_00129

[SWS_WdgM_00377][

Error Name:	WDGM E_IMPROPER_CALLER	
Short Description:	Defensive behavior checks have detected an improper caller.	
Long Description:	This extended production error indicates that the mode switch request has been invoked with a wrong caller id.	
Detection Criteria:	Fail	Improper caller
	Pass	No improper caller
Secondary Parameters:	-	
Time Required:	detected immediately when mode request is invoked with a wrong caller id	
Monitor Frequency	Aperiodic - the detection (supervision) occurs only if the mode switch is triggered.	

图9.10　AUTOSAR规范中对防御性的设计方法的应用案例

除此之外，为了确保按照AUTOSAR规范编写的代码安全，AUTOSAR组织还邀请PRQA公司成为合作伙伴，一起研究“安全系统中C++14语言的使用指南”，并作为Adaptive AUTOSAR产品标准的准则一起发布。

9.2 AUTOSAR中实现FFI的安全机制

9.2.1 AUTOSAR安全机制的存储空间分区

ISO 26262 Part 6中对存储空间分区（Memory Partition）的要求为如果软件分区是用于实现软件组件间的FFI，则应确保以下内容。

（1）共享资源使用时需利用软件分区来确保FFI

共享资源使用时需利用软件分区来确保FFI，这意味着：

①在软件分区内的任务不会彼此之间相互干扰；

②一个软件分区不能更改其他软件分区的代码或数据，也不能控制其他分区上的非共享资源；

③来自于一个分区的共享资源服务不能受另一个软件分区的影响，这包括在访问资源时对性能和调度延时等方面造成的负面影响。

（2）软件分区由专用硬件特性或等效手段予以支持（适用于ASIL D）

（3）实现软件分区的软件应遵循相同或高于最高ASIL等级要求

为实现上述功能安全对分区隔离的要求，AUTOSAR提供了存储空间分区（Memory Partition）机制。存储空间分区可将两个软件组件映射到不同的存储空间上，从而避免它们之间的相互干扰，即在一个SWC上存储空间的失效，不会引起另一个SWC的崩溃。

因为有了不同的ASIL等级，所以就有了不同ASIL等级之间需要隔离这个概念，需要避免低等级的任务修改高等级任务的数据，造成高等

级任务使用了不可信的数据，从而影响安全功能。在软件架构设计（Software Architectural Design）阶段，完成静态架构和动态架构设计之后，需要做软件独立性分析，这里重点就是分析这部分内容，分析结果有可能会影响不同模块ASIL等级的调整。

软件存储分区是OS及RTE功能的扩展，是通过OS中的OS-Application来实现的。OS-Application可用作一个独立的错误控制区域，将SWC和相关资源可映射到OS-Application上。将几个SWC组成一个分组并将这些分组运行于不同的存储区间上，从而避免它们之间的相互影响。通过这种方式能够确保某一个SWC相关的存储空间错误不会影响到其他的SWC，一旦检测到OS-Application错误，应用程序可以在运行时终止或重新启动，同时也可为每一个OS-Application单独设定恢复（Recovery）策略。

图9.11展现了一个分区的示例。假设在Partition 1中发生了某个错误/冲突，那么Partition 1会被OS的服务函数终止，同时Partition 1上的通信也被停止，然后Partition 1重新启动。

图9.11　软件存储分区示例

9.2.2　AUTOSAR安全机制的存储空间保护

在AUTOSAR OS SC3/SC4中提供了存储空间保护（Memory Protection）特性。存储空间保护主要是避免一个软件组件在未经许可的前提下篡改另一块存储空间的软件组件的数据内容。为此，在AUTOSAR中定义了OS-Application的可信度，分为可信（Trusted）的和不可信（Non-Trusted）的两类，在同一个OS-Application内的OS对象之间可以相互访问而不受限制，而不同OS-Application之间的OS对象之间的访问是需要授权的。在AUTOSAR OS中每个Application只允许可信（Trusted）的应用访问，它们在运行时，对存储及OS API的访问不受限制，不需要运行时的时间保护。而对于非可信（Non-Trusted）的OS-Application，如果监控或保护没有开启的时候不允许运行，它们受限访

问存储、OS API等。这样就可以防止非可信的另一个软件组件对应用的破坏。

存储空间的访问控制依赖于OS的存储保护模块，需要提醒的是这一功能需要硬件拥有存储保护单元（Memory Protection Unit，MPU）特性的支持，即硬件和软件的结合才能起到存储空间访问的保护控制。这种机制确保ECU上安全相关的应用对存储保护的需求，防止一个非授权的软件组件对数据的破坏。存储保护针对不同的数据段，代码段和任务栈有不同的实现机理，如图9.12所示。

图9.12　存储空间保护示意

（1）对数据段的保护机制

OS-Application支持私有数据段，任务或中断处理程序也可以有私有数据段，OS-Application的私有数据段可以被从属于这个OS-Application的所有任务和中断处理程序共享。操作系统禁止非可信OS-Application对自己的数据段的写操作，并禁止对另一个OS-Application的数据段的读（可选的）/写操作，但允许对自己的私有数据段的读/写操作。

（2）对代码段的保护机制

代码段既可以被某个OS-Application私有，也可以被所有的OS-Application共享。操作系统保护代码段可选的不可被非可信的OS-Application执行。

（3）对任务栈的保护机制

不同的OS-Application内的任务和OS中断处理程序的任务栈数据是不需要共享的，因此不同OS-Application之间的栈需要受到保护。操作系统禁止非可信OS-Application对自己的任务栈的写操作，允许任务或者二类中断处理程序对私有任务栈的读写操作，但是禁止非可信OS-Application内的其他任务/中断对私有任务栈的写操作。

9.2.3 AUTOSAR安全机制的程序流监控

程序流监控（Program Flow Monitor）作为ISO 26262中强烈推荐的一项，对于电控软件的安全提升有着举足轻重的意义。通过程序流监控，能够发现软件运行过程中的一些违反设计意图的错误，从而能够进行相应的应对措施，确保行车安全。

在AUTOSAR规范所定义的软件架构中，程序流监控功能的实现主要由“看门狗”栈（Watchdog Stack）来实现，自上而下包括WdgM模块（Watchdog Manager）、WdgIf模块（Watchdog Interface）和Wdg驱动模块（Watchdog Driver）。对应用层来说，这三个模块通过RTE向应用层提供接口服务，由此实现底层监控应用层策略运行的功能。

下面先分别介绍一下这三个模块，引入一些基本概念，方便后续针对程序监控这一安全机制的介绍。

（1）WdgM模块

WdgM模块位于AUTOSAR架构基础软件层中的服务层，如图9.13所示。WdgM主要负责监控程序执行的准确性，并触发相应的硬件“看门狗”，即所谓的“喂狗”，扮演了整个监控的核心角色。WdgM监控的对象为监控实体（Supervised Entity，SE），每一个监控实体均会映射至对应的OS Application。由于OS Application无法涵盖跨核的Task，因此一个监控实体无法实现跨核任务的监控，需多个监控实体配合实现。

图9.13 AUTOSAR“看门狗”栈架构

Supervised Entity中重要的一些位置被定义为检查点（Checkpoint，CP），也是WdgM模块获取Supervised Entity实际运行状态最为核心的元素。根据设计者的设计意图，一个Supervised Entity的执行是有相应执行顺序的。因此，其内部的若干Checkpoints也有相应的转移顺序，在AUTOSAR的定义中，这一转移称为Transition。根据Checkpoint是否存

在于同一个Supervision Entity可分为Internal Transition以及External Transition。Checkpoints及其Transition形成了监控实体运行的逻辑顺序图，称为Graph。同样，根据是否包含External Transition，Graph也可分为Internal graph以及External Graph。Graph作为WdgM实际监控程序执行逻辑的依据，在实际应用中，其设定需要确保准确无误。上述提及的相关概念示意如图9.14所示。

图9.14　Supervised Entity相关概念示意

WdgM提供了三种监控方式，包括Alive Supervision、Deadline Supervision以及Logical Supervision。三种监控方式针对不同的监控应用场景。Alive Supervision用于监控周期性任务是否被执行到，即该任务是否还“活着”；Deadline Supervision用于监控任务是否在指定

的时间内按时完成；Logical Supervision用于监控任务是否按照设计的顺序得到正确的执行，是监控嵌入式系统软件是否正确执行的基本技术。

（2）WdgIf模块

WdgIf模块作为整个Watchdog Stack的一部分，其主要功能是连接上层WdgM模块与底层Watchdog驱动模块。所连接的底层Watchdog驱动模块可以是一个，如外部“看门狗”；也可以是多个，如CPU每个核的内部看门狗。

针对多核系统的监控设计，可以通过WdgIf模块实现连接各个核上的WdgM Instance及“看门狗”驱动。目前包含两种实现途径。

一种是每个核的WdgM Instance都被WdgIf连接至其对应核上的CPU Watchdog，不同的核互不干扰，如图9.15所示。当一个核上出现了故障，经WdgM监测确认后，通过WdgIf对该核上的Internal Watchdog进行相关操作，如取消“喂狗”。

图9.15　多核系统的监控设计（方法1）

另一种是通过带有State Combiner特性的WdgIf功能来实现，不同核上的WdgM Instance共享一个“看门狗”（外部“看门狗”），如图9.16所示。

图9.16　多核系统的监控设计（方法2）

State　Combiner有主从之分，主State　Combiner对应的WdgIf连接外部“看门狗”驱动。其余的从State　Combiner接收到各自核上WdgM　Instance监控结果后，将相关信息更新至一块共享的内存区间。从State　Combiner对该段内存进行写操作，主State　Combiner则进行读操作，由此将不同核上WdgM监控的结果统一汇总至主State Combiner对应的WdgIf，从而决定是否正常“喂狗”。

（3）Wdg模块

Wdg模块，即Watchdog驱动模块，提供了三类服务。

①初始化“看门狗”的服务。

②改变“看门狗”运行模式，包括Fast、Slow、Off三种。需要注意的是，根据AUTOSAR规范，Slow模式只被建议用于系统启动初始化过程

中，并且对于安全系统，OFF模式不建议使用。

③触发“看门狗”，即“喂狗”操作。

如上所述，Checkpoints标志了程序执行过程中的重要阶段。当程序运行到Checkpoint处，监控实体Supervised Entity会直接调用函数WdgM_CheckpointReached（），或经RTE封装的对应函数。设计者在配置WdgM模块的时候，通过Checkpoint以及Transition生成期望的程序运行逻辑，因此WdgM能够实时监控程序执行是否符合预期。需要注意的一点，Checkpoints设置得越多，越能准确地反映程序代码执行的潜在可能。自然地，这会额外消耗部分芯片资源（Memory、CPU Load等），弱化控制器的性能。

AUTOSAR规范中定义的WdgM模块程序监控机制运作流程示意如图9.17所示。WdgM_CheckpointReached（）函数执行是在监控实体上下文内实现的，而WdgM_MainFunction（）主函数是在OS Scheduler上下文内执行的。

图9.17　WdgM模块程序监控机制运作流程示意

三类监控功能会对每个监控实体生成一个监控正确与否的状态，每个监控实体有三个维度的状态，包括Alive Supervision、Deadline Supervision以及Logical Supervision。基于这些在WdgM初始化的过程中，所有的状态都会被赋为Correct。实际监控过程中，一旦发现

错误，就会将对应状态置为Incorrect。由此每个监控实体都有一个监控正确与否的状态，AUTOSAR规范中定义为Local Status。其中对于Deadline Supervision以及Logical Supervision，其监控实体的Local Status是在WdgM_CheckpointReached（）函数执行过程中被确认的，而对于Alive Supervision，其监控实体的Local Status是在WdgM_MainFunction（）主函数内被确认的。所有监控实体的Local Status汇总成WdgM Global Status。最终，WdgM_MainFunction（）函数就是依据Global Status来决定后续是否正常触发“看门狗”。

下面对Alive Supervision、Deadline Supervision以及Logical Supervision三种检测方式进行具体分析。

（1）Alive Supervision

Alive Supervision主要用于检测WdgM_CheckpointReached（）函数被调用的频率是否符合设计需求。对于一个周期性的监控实体Supervision Entity，需要设定一个Checkpoint，在该监控实体周期性运行的时候，正常情况下，WdgM_CheckpointReached（）函数也能得到周期性的调用。这里引入几个概念。

①Supervision Reference Cycle：定义的Alive Supervision监控参考周期，一般是WdgM主函数周期的整数倍。

②Expected Alive Indications：WdgM_CheckpointReached（）函数期望的调用次数。

③Min/Max Margin：期望的WdgM_CheckpointReached（）函数调用次数的上下偏差。

在监测参考周期内，许可的WdgM_CheckpointReached（）函数调用次数应在［Expected Alive Indications-Min Margin，

Expected Alive Indications+Max Margin］范围内。实际监控过程中，如果WdgM_CheckpointReached（）函数的调用次数超出了许可的范围，则说明该段监测参考周期内，监测实体对应的任务执行过快或者出现未执行到的情况，相应地会报出监控错误。

下面列举两个用例，分析上述参数设置及其监测的可能结果，以及参数设置对于监控效果的影响。WdgM主函数运行周期为20ms，任务周期为30ms，用例A的监测参考周期为20ms，用例B的监测参考周期为40ms，主要参数设置如表9.1所示。

表9.1　Alive Supervision用例主要参数设置

参数	用例 A	用例 B
Expected Alive Indications	1	2
Supervision Reference Cycle	1	2
Min Margin	1	1
Max Margin	0	0

如图9.18所示，用例A时在每个监控参考周期内，WdgM_CheckpointReached（）函数调用次数为0或者1。若以此配置，则当该监控实体不再执行时，仍满足许可的调用次数范围，从而不能有效监控潜在的错误。而对于用例B，WdgM_CheckpointReached（）函数调用次数为1或者2，许可的调用次数范围［1，2］，因此一旦监控实体不再执行，则超出许可范围，能够检测出错误，但是仍有可能出现监控实体偶发性未被执行却无法监测出来的情况。因此在实际配置过程中，需要谨慎选择WdgM主函数周期、监控实体任务周期以及监控参考周期，确保不会出现上述两种情况发生。

图9.18　Alive Supervision用例示意

一般设计者可以选取WdgM主函数周期及监控实体任务周期两者的最小公倍数来设置监控参考周期，确保每个监控参考周期内的WdgM_CheckpointReached（）函数调用次数都是固定值，此时，上下偏差Min/Max Margin均设为0。

（2）Deadline Supervision

Deadline Supervision对Transition执行时间进行监控，也就是对监控实体执行的动态行为进行监控。涉及的主要参数为Deadline时间上下限，若该Transition无法在Deadline时间上下限内完成，则监测出错误。对于应用层来说，主要是监控相邻的两个WdgM_CheckpointReached（）函数之间的代码执行时间。由于预测一段代码的执行时间比较困难，一般情况下很难给出比较精确、有意义的Deadline时间上下限，因此相比于另外两类监控，Deadline Supervision的应用场景较少。

如图9.19所示，当代码执行至Checkpoint处时，执行WdgM_CheckpointReached（）函数，在该函数内，会计算当前OS-tick计数值。同时，与上一个WdgM_CheckpointReached（）计算的OS-tick值相减，由此得到两个Checkpoint之间Transition所经历的OS-tick时间，并与配置得到的时间上下限进行比较，若超限，则报出该监控实体Deadline Supervision状态错误。

图9.19 Deadline Supervision机制示意

（3）Logical Supervision

对于Logical Supervision，即程序流监控（Program Flow Supervision），是功能安全标准ISO 26262中极力推荐的，用于检测程序代码的执行逻辑是否正确。这里先引入两个概念。

①Program Flow Reference Cycle：程序流监控参考循环，也是对应监控实体的实际运行周期，一般以WdgM监测主函数周期的整数倍来定义。

②Failed Program Flow Reference Cycle Tolerance：程序流监控基准循环内可接收的错误数量，对于安全系统，一般该参数设为0。

在进行WdgM配置的过程中，形成了逻辑流图Graph。如图9.20所示，一旦程序实际执行过程中，运行至某Checkpoint处，则调用WdgM_CheckpointReached（）函数。在该函数体内，会记录上一个Checkpoint Id以及两个Checkpoint之间的Transition Id（Transition Id根据相邻两Checkpoint计算得到），若该Transition不属于配置的Transition Group，也就是说该相邻Checkpoint之间的Transition不属于预期Transition，则报出该监控实体Logical Supervision状态错误。

图9.20　Logical Supervision机制示意

9.2.4　AUTOSAR安全机制的E2E保护

在ISO　26262：Part　6附录D中描述了信息在交换过程中面临的风险，如：

①重复接收；

②信息丢失；

③信息延迟；

④非预期插入信息；

⑤信息伪装或信息的不正确寻址；

⑥信息序列错误；

⑦信息损坏；

⑧一个发送端给多个接收端的信息不对称；

⑨接收方仅接收到部分子集信息；

⑩通信信道访问阻塞。

在信息的传输链中，任何一个节点发生问题，都可能导致最终的信息不正确（图9.21中各箭头所示），如下所示。

图9.21　AUTOSAR E2E保护机制

①硬件问题：如CAN收发器的寄存器损坏。

②外部干扰：如电磁干扰。

③软件本身错误：如RTE、Com、IOC或OS。

因此，功能安全要求在与安全相关的汽车系统中软件有一定的机制来检测类似的问题，使用安全数据传输来保护组件之间的通信，以确保交换数据的正确性。这意味着：

①应防止通信错误（如通过适当的软件架构和核查手段来确保）；

②如果仅有差错预防还不够（如对于ECU之间的通信），应在ECU实时运行时能够检测到足够多的错误，而未发现的危险错误率应该低于某些允许的限值。

AUTOSAR中的E2E通信保护机制通过在信息中增加额外的Counter和CRC机制能够监测出的通信问题如下。

①重复接收：同一帧被多次重复接收。

②信息丢失：帧或帧中的部分信息丢失。

③信息延迟：违反帧的时序要求。

④非预期插入信息：传输非预期帧或帧中添加了非预期内容。

⑤地址错误：帧发送给了错误的目标地址。

⑥信息序列错误：帧未按正确序列接收。

⑦信息损坏：帧或帧的部分内容被篡改。

⑧一个发送端给多个接收端的信息不对称。

⑨信息伪装：接收非授权的信息，该未授权的信息被伪装成了是正确节点发出的信息。

根据不同的E2E库（E2E Lib）使用方式，可大致分为以下几类。

（1）E2E保护包装器

E2E保护包装器（E2E Protection Wrapper）是一种在ASW层实现的数据保护机制。用户在RTE之上进行一次封装，E2E Protection Wrapper调用E2E Lib提供的函数库，实现E2E的保护和校验，最终调用RTE的API进行发送和接收。这种实现方式适用于不同层次软件组件之间的通信，小到同一个核上的SWC之间的通信，大到跨ECU SWC之间的通信都是适用的。

（2）COM E2E Callout

COM E2E Callout是一种非标准的保护I-PDU信号的集成代码，是针对跨ECU之间的通信。COM E2E Callout的E2E保护和校验是在基础软件层做的，在这种实现方式下检验的单元是以PDU的形式存在的，某一个PDU发送或者接收时，会触发该PDU的Callout函数，在Callout函数内部实现E2E的保护和校验。

（3）E2E保护包装器与COM E2E Callout混合使用

E2E保护包装器与COM E2E Callout混合使用方式较为灵活。例如：对于特定的数据元素，发送方使用COM E2E Callouts，而接收方采用E2E保护包装器；或在给定的ECU网络或一个ECU中，一些数据元素受E2E保护包装器的保护，而另一些则使用COM E2E Callouts。

（4）E2E Transformer

E2E Transformer是一种新的，在RTE级别实现的保护方式。RTE会调用E2E Transformer的API，E2E Transformer的API进一步调用E2E Lib提供的函数库，实现E2E的保护和校验。所有的函数调用全部封装在RTE内部实现，这是AUTOSAR规范要求的，但是目前多数主流的AUTOSAR工具链供应商并没有实现这种方式，也有AUTOSAR工具链供应商已经实现了E2E Transformer，比如ETAS。

下面将详细地阐述如何在项目中实现E2E Protection Wrapper，其他几种方式的实现原理类似于此，可参考AUTOSAR技术规范有关章节。

在这种方法中，每个与安全相关的SWC都有自己的附加子层，称为E2E保护包装器，负责将复杂数据元素编组到与相应的I-PDU相同的布局中（用于ECU之间的通信），以便为正确地调用E2E库和RTE做好准备。

E2E保护包装器的使用使得无须采取更多措施，即可确保SWC之间的VFB通信的完整性。既可以是同一ECU内的SWC间通信（意味在同一或不同的核或在一个微控制器的同一个或不同的Memory分区），也可以是跨ECU间的SWC通信（SWC连接由VFB实现）。

E2E保护包装器是SWC间通信的系统解决方案，其不太关心使用何种资源（如COM和网络、OS/IOC或RTE内部通信）。对传统的应用代码或模型（SWC），基本上无须更改原先的程序代码，只需在接口中增加少许保护参数即可。

需要提请读者注意的是，E2E保护包装器不支持SWC的多个实例化。这意味着，如果一个SWC应该使用E2E保护包装器，那么这个SWC必须是单实例化的。

所谓包装器，实际是封装了SWC的接口函数，如Rte_Write和Rte_Read函数，使用E2E库对Rte_Read/Write进行调用，同时进行数据交换保护。对于要传输的数据元素，有一组为发送方和接收方生成的包装函数（读/写/初始化）功能，如E2EPW_Write_<p>_<o>（）和E2EPW_Read_<p>_<o>（），其返回值为表示状态的32位整数。

下面给出一个简单的例子。在没有E2E保护的时候，从发送方到接收方主要过程如下：两个SWC之间通过Sender-Receiver接口传递一个DataElement DE1，里面有两个安全信号，即s1和s2，其数据结构假设如下。

```
DE1_Type
{
  U2 s1;
  U8 s2;
}appdata;
```

在发送时，直接调用RTE的接口函数Rte_Write_XXX即可。

若采用Profile 2的E2E保护（详细的Profile 2结构请参见有关规范），如图9.22展示了E2E库和E2E保护包装器从发送到接收的总体使用流程（标签上的第一个数字定义了执行顺序）。

图9.22　基于Profile 2 E2E保护的数据传输过程示意

此时，原先的数据结构会进行一定修改，变成一个新的数据结构，如下所示。

```
E2E_DE1_Type
{
  U8 crc;                    /*新增的CRC值*/
  U8 counter;                 /*新增的计数器*/
  U16 s1;
  U8 s2;
}e2e_appdata;
```

对于发送方而言，首先，应用层会调用一个E2EPW_Write_<p>_<o>（图9.22中步骤S1，下同），在这个函数里初步处理后先调用E2E_P02_Protect（S4）函数。需要注意的是，这里要先完成从普通结构体类型，如E2E_DE1_Type，到一个规则u8数组类型的形式转换（S0）。譬如：针对上述E2E_DE1_Type数据结构，将定义一个u8 E2E_DE1_Array［5］的数组，并将原数据结构中的元素进行相应转换，如下所示。

```
Array［0］=crc；
Array［1］=counter；
Array［2］=s1≫0x08；
Array［3］=s1&0xff；
Array［4］=s2；
```

这是很好理解的。因为所谓Protect，就是计算CRC值后调用E2E Lib，而CRC的计算需要的对象是u8类型的数组（如这里的E2E_DE1_Array），而不可能是任一类型的数据（如这里的E2E_DE1_Type）。

然后，再调用Rte_Write_<p>_<o>_（Instance，E2E_DE1_Array）（S7），即完成了数组的发送。

对接收方而言，整个过程是完全相反的。接收方会先调用一个E2EPW_Read_<p>_<o>的函数（R1）。在这里面，先把数据以数组形式接收过来，即调用Rte_Read_<p>_<o>_（Instance，E2E_DE1_Array）（R2）；然后再检查数据是否有问题，即调用E2E_P02_Check（R4）；如果没有任何问题，再从数组中还原原始数据（R8）。

在实际项目开发过程中，通常需要E2E保护的信号散落在各个SWC中，为方便统一管理，通常建议使用以下设计模式（图9.23），即把安全信号集中后统一经Conversion做转换，再经Manager模块后统一管理发

送和接收的信号。

图9.23　安全信号集中管理的软件架构示意

9.3　本章小结

本章主要介绍了AUTOSAR对功能安全的相关支持情况。首先，从ISO　26262标准着手，介绍了功能安全中对软硬件设计方面的诸多指导性建议，并关联到AUTOSAR规范，介绍了AUTOSAR中对这些安全相关需求的支持情况。之后，着重对AUTOSAR规范中为实现功能安全FFI目标所提出的安全机制进行了详细讲解。通过本章的学习，可以从宏观上理清AUTOSAR与ISO 26262之间的关联，给读者一个全局的概念性认识，从而可以更好地理解两者之间的联系。

第10章　AUTOSAR技术展望

经过10多年的发展，AUTOSAR已广泛应用于众多领域控制器中，其在汽车电子领域与其他领域技术发生了许多交叉。同时，随着越来越多的应用领域扩展，新技术的应用也推动了AUTOSAR向新的方向去发展。本章介绍AUTOSAR对信息安全方面的支持，以及新一代Adaptive AUTOSAR平台的情况。

10.1 AUTOSAR与信息安全

首先，在此稍微澄清一下功能安全和信息安全两者之间的关系。功能安全（Functional Safety）指的是该系统不会对人身安全、财产以及环境造成伤害。而信息安全（Cyber Security）则是指安全系统难以被其他人恶意利用车辆漏洞导致经济损失、驾驶操控失误、隐私盗取及功能安全的损坏，如车速不受到远程控制等。因此，任何功能安全的系统本身都是信息安全系统，反之则不然。例如，对一个扭矩控制系统而言，假如该系统软件存在安全漏洞，且被攻击者利用，那么在驾驶过程中就有可能会因被破解实现远程控制后而导致出现人身安全风险，即该功能安全系统必然属于信息安全系统。而如存在信息安全漏洞的车载娱乐信息系统，即使直接对其进行安全攻击，也并非一定会引起直接的驾驶功能破坏及人身安全损失，或许只是会引起部分私人信息的泄漏等，并不会导致功能安全风险。

汽车单纯地作为通勤工具，能够听听广播、放放CD就以为是多么了不起的“多媒体”时代一去不复返了。现如今，汽车可谓是内外连通：不仅能与车内设备连接，同时还可以方便快捷地接入互联服务。而目前人们最担心的，正是这些以直接或者间接方式连接了互联网的汽车很可能暴露于恶意软件的代码和数据攻击之下，会导致类似SRS等重要安全系统的失灵，造成巨大的危害。

但毋庸置疑，长期以来汽车一直注重系统功能安全的可靠性，并在不断增强车辆主被动安全能力。在功能性安全问题上一般只需考虑开发人员可能发生的疏漏，即功能性安全隐患一般源于系统故障、软件或硬件失效。但在信息安全领域，还必须同时考虑恶意或意外行为可能造成的影响，包括黑客乃至车主的行为。比如有些车主出于好奇，也有可能对车辆进行一些不当操作，从而影响车辆的网络安全。

对工程师而言，过去需要关注的仅仅是车辆硬件和软件之间的配合，而如今他们还要考虑更多的问题，比如外来入侵者是否有可能通过某些方法影响车辆的关键功能，如车速控制等。因此汽车生产商必须采取系统工程的方法保护系统安全，必须及时拿出预防措施，在车辆的整个生命周期中为其提供有效保护。必须开发出全面的安全保障策略，有效应对常规问题，并对各种恶意攻击做出敏捷的反应，以确保车辆不会成为黑客攻击的受害者。

当前常规的网络安全系统一般采用纵深防御（Defense in Depth）技术，这样一来即使某层防御被突破，其他程序也能补上缺口。此外，分层防御还能保证问题发生时可以得到有效控制，不会迅速蔓延至车辆的其他系统。本章节讨论的车辆端的网络安全，更多从车辆的角度看，主要包括：车联安全和车内安全。

（1）车联安全

车联安全包括：

①安卓应用下载；

②远程ECU固件更新OTA（Over-the-Air Technology）；

③政府部门和保险部门用的“黑盒子”；

④车间通信CAR-TO-CAR等。

（2）车内安全

车内安全包括：

①防盗/零部件保护；

②里程保护；

③发动机特性；

④乘客数据；

⑤安全启动和信任链；

⑥安全通信等。

图10.1展示了NXP公司所提出的多层汽车安全等级概念。这个四层安全等级架构展示了一个汽车系统通信、硬件和软件组件的通用概念。

图10.1　NXP提出的多层汽车安全等级概念

（1）Secure Interface（Level 1）

Level 1定义了汽车与外部接口的安全方案。主控制器集成了众多硬件和软件方面的安全技术，如安全验证、安全密匙存储等。

（2）Secure Gateway（Level 2）

Level 2定义了外部环境和车内网络系统间接口的网络安全隔离方案。初始化安全网关后激活域隔离、防火墙、滤波器和中央入侵探测IDS等功能。防火墙提供了对车内网络的安全保护功能，防止外部恶意攻击。

（3）Secure Network（Level 3）

Level 3提供了车内通信间的通信保护，如CAN/CANFD/Ethernet等。其技术有CAN Id Killer、消息验证、分散入侵探测IDS等。

（4）Secure Processing（Level 4）

Level 4则需要集成硬件支持保护每个ECU的执行环境。这些硬件所支持的安全执行元素主要有硬件安全模块（Hardware Security Module，HSM）或称为可信执行模块（Trusted Platform Module，TPMs）。Level 4提供了安全启动、ECU上运行软件完整性保证、OTA等特性。

如前一章对功能安全的介绍，本章不会详细描述信息安全相关的开发流程、组织体系、信息安全概念以及要达到某个信息安全等级所要采取的各种安全方案，如安全启动策略、安全刷写、安全分区等，而只就当前AUTOSAR R4.x中BSW直接支持信息安全有关的协议栈模块作一个简单的介绍。

在当前的AUTOSAR R4.x规范中，与信息安全相关的主要有密码抽象库（Crypto Abstraction Library，包括CAL和CPL两个模块）、密码协议栈（Crypto Stack，包括CSM、CryIf和Cry三个模块）以及安全车载通信（Secure Onboard Communication，SecOC）三部分，支持对应四层防

御体系中不同的Level。上述各模块在AUTOSAR分层架构中所处的位置如图10.2所示。

图10.2　信息安全相关模块在AUTOSAR分层架构中的位置

图10.2中两路不同的密码服务路径可以简单地用CAL和CSM来代表。两者功能非常类似，都提供了密码算法（Cryptographic Primitives，密码原语或者称为加密基元）。从架构上说，两者其实都由封装接口层和密码实现层构成。加密算法库CPL（Cryptographic Primitive Library）和加密库模块CRY（Cryptographic Library Module）即是具体的实现层。

由于密码抽象库和密码协议栈是两条并行的加密路线，两者可选其一，故后续小节主要选密码协议栈和安全车载通信作进一步的介绍。

10.1.1　密码协议栈

AUTOSAR密码协议栈功能主要包括哈希值计算、非对称签名验证和对称数据加密等，其架构如图10.3所示。在AUTOSAR BSW中从上到

下它主要包括以下三个模块：

图10.3　AUTOSAR密码协议栈架构

①密码驱动模块（Crypto Driver，Cry）；

②密码接口模块（Crypto Interface，CryIf）；

③密码服务管理模块（Crypto Service Manager，Csm）。

（1）Cry模块

Cry模块位于MCAL层，它实现了具体的硬件（如SHE）或者软件驱动，即可以使用软件或硬件密码算法，管理来自于不同应用的加密服务请求。这两种算法各自的劣势在于：硬件密码算法严重依赖于目标硬件的支持，而软件密码算法则无安全密钥存储（软件密码算法采用NvM管理密钥存储）。

Cry模块中支持的密码算法按大类可分为对称的和非对称的两类。非对称的有RSA2048、RSA4096、ECC256、ECC512等，对称的则有AES等。具体支持的算法标准需参见供应商的用户手册。需要提醒的是，不管是硬件算法还是软件算法，两者的具体实现都没有在AUTOSAR规范中定义。

（2）CryIf模块

Cry模块之上的CryIf模块则抽象了硬件算法与软件算法等差异，提供了通用接口给CSM。这样一来，对应用层而言就完全不需要关心密码算法是硬件实现还是软件实现的，以及具体采用哪个加密算法等特性。

（3）Csm模块

对应用层而言，Csm模块经RTE提供标准的密码服务接口，而对BSW或复杂驱动而言，直接调用相关API即可。Csm模块可支持不同的应用使用同一个密码服务，而其密码算法可以完全不同，例如：一个应用需要SHA-2，而另一个则需ECC512。它同时支持异步调用，即加密/解密完成后经Callback函数告知应用层。

Csm模块按优先级排队控制着一个或多个客户端对称或非对称密码服务的并行访问，提供的服务主要包括：

①哈希值计算；

②信息真实值（Message Authentication Code，MAC）的产生和验证；

③数字签名的产生和验证；

④对称和非对称加密/解密；

⑤随机数的产生；

⑥安全计数器；

⑦密钥管理，如密钥的产生和设置等。

Csm模块中众多服务可采用顺序流服务（Streaming Services）或者单个函数调用（Single Call）的方法。所谓顺序流服务就是可以把一个具体的任务（如随机数的产生）拆分为三个进程：Start、Update和Finish，如图10.4所示。

图10.4　顺序流服务示意

顺序流服务具体过程如下：

①首先，调用Csm_SymEncryptStart通知开始一个新的特殊任务或初始化某个密码计算；

②其次，将输入数据提供给Update更新函数，如

Csm_SymEncryptUpdate，计算出一个中间结果；

③最后，调用Finish结束函数，如Csm_SymEncryptFinish，指示密码计算已完成。

为提高密码服务的性能，可以把几个操作合并成一个函数调用，如这个例子中直接调用一个函数Csm_GenerateRandom即可直接生成随机数。

上面提到的硬件加密算法依赖于硬件安全模块——安全硬件扩展（Secure Hardware Extention，SHE），如图10.5所示。它主要有以下特点：

图10.5　安全硬件扩展结构示意

①独立的存储空间，可存储关键安全数据；

②AES加密/解密引擎；

③哈希算法；

④伪随机数产生器；

⑤密钥管理；

⑥MAC产生和验证；

⑦安全启动；

⑧安全时钟等。

由上述特点可见，SHE支持安全地生成、存储和处理安全性关键材料，屏蔽任何潜在的恶意软件，运用有效的篡改保护措施限制硬件篡改攻击的可能性，通过应用专门的加密硬件保护软件的安全性。另外，相比于软件加密服务，硬件速度性能大概快几十倍，能更快地响应服务请求者的服务。这些种种优异特性使得SHE在高等级的信息安全需求中作为值得信赖的安全锚显得必不可少。

这里，SHE是一个通用的名字，不同的供应商可能有不同的命名，如NXP称其为密码服务引擎（Cryptographics Service Engine，CSE）。有些芯片上还有功能更灵活的硬件加密模块，如英飞凌Aurix系列里有硬件安全模块（Hardware Security Module，HSM）。区别在于HSM支持用户自己写加密算法，有防火墙功能等，显得更强大。

10.1.2 安全车载通信

对敏感数据的身份验证和完整性保护是保护车辆系统的正确和安全功能所必需的，这可以确保接收到的数据来自正确的ECU，并具有正确的值。AUTOSAR规范中引入安全车载通信
SecOC（Secure Onboard Communication）安全机制，可以在汽车嵌入式网络的两个甚至多个节点之间传输安全加密数据，能够防止信号的随机错误、非法注入、故意更改、复制回放等攻击，旨在为

PDU（Protocol Data Unit）的关键数据提供有效可行的授权和认证机制，实现安全的通信功能。

AUTOSAR安全车载通信过程示意如图10.6所示。在此示意图中，PduR模块负责将传入和传出的与安全相关的I-PDU路由到SecOC模块。然后，SecOC模块会添加或处理安全相关信息，并将结果以I-PDU的形式传播回PduR模块。PduR模块负责进一步路由新的I-PDU。SecOC模块及PduR模块所支持的各种通信范式和原则，特别是多播通信、传输协议和PduR网关。但是，由于SecOC模块仅限于向上层的API，因此当前不支持从/到DCM与J1939DCM对PDU进行身份验证。

图10.6　AUTOSAR安全车载通信过程示意

在AUTOSAR的分层架构中，SecOC模块被设计在PduR模块旁边，它和PduR模块交互过程示意如图10.7所示。

图10.7　SecOC模块与PduR模块交互过程示意

图10.7中同时展现了两个方向的数据流。

（1）从上至下——信息“授权”

从上至下的原始“真实I-PDU”（Authentic I-PDU）到带有安全信息的“安全I-PDU”（Secured I-PDU）的过程，可称为信息“授权”（Authentication，可类比于“加密”）。即对发送方来说，PduR模块从Com模块接收到需要SecOC保护的原始信息，在发送给底层之前，增加合适的授权信息后成为受保护的“安全I-PDU”。

（2）从下至上——信息“验证”

从下至上的带有授权信息的“安全I-PDU”还原为原始“真实I-PDU”的过程，可称为信息“验证”（Verification，可类比于“解密”）。即对接收方来说，接收到底层传上来的I-PDU后，PduR模块调用SecOC验证来自底层的信息，将原先含授权信息的I-PDU信息验证后再送给Com模块处理。

在进一步解释如何进行信息授权和信息验证前，先解释一下“真实I-PDU”及“安全I-PDU”等相关概念。所谓“真实I-PDU”是指原始的AUTOSAR I-PDU，对其需要进行保护，以防止未经授权的操作和重播攻击。而所谓“安全I-PDU”则是由在“真实I-PDU”中添加额外的授权信息（Authentication Information，如消息验授权码MAC等）构成，如图10.8所示。

图10.8　安全I-PDU组成示意

图10.8中授权码Authenticator（所谓MAC）是指使用密钥Key、“安全I-PDU”的数据标识符DataId、“真实I-PDU”和新鲜度值（Freshness Value）等经过某个生成器生成的唯一身份验证数据字符串。MAC提供了一个很高的可信度，即“真实I-PDU”中的数据是由合法源生成的，并在其预期的时间提供给接收ECU。这里，新鲜度值是指用于确保“安全I-PDU”的新鲜性的单调计数器。这种单调计数器可以通过单个的消息计数器（称为“新鲜计数器”）来实现，或者用一个称为“新鲜时间戳”的时间戳值。需要注意的是，这里DataId是表征原始PDU的代号，发送和接收方需相同。最终，在“安全PDU”中，MAC和Freshness的值可以只截取其中一部分，并不一定是全部的原值。

下面将对信息“授权”以及信息“验证”过程进行详细介绍。

（1）信息“授权”过程

通常，SecOC模块与PduR模块交互授权“真实I-PDU”的主要步骤如下。

①对于“真实I-PDU”的每个传输请求，上层COM模块应通过PduR模块传输来调用PduR_Transmit。

②PduR模块将此请求路由到SecOC模块，并调用SecOC_Transmit。

③SecOC模块将“真实I-PDU”复制到自己的内存并返回。

④在其MainFunction的下一次预定调用期间，SecOC模块通过计算身份验证信息并通过PduR模块通知相应的较低层模块来创建“安全I-PDU”。详细步骤如下。

a.准备缓冲区。在准备过程中，SecOC应分配必要的缓冲区以保存身份验证过程的中间和最终结果。

b.准备给验证器的数据。SecOC模块应构造DataToAuthenticator，即用于计算验证器的数据。DataToAuthenticator是通过将DataId、完整的“真实I-PDU”和完整的新鲜度值Freashness串联起来形成的。即：

DataToAuthenticator=数据标识符DataId|真实I-PDU|完整的新鲜度值Freashness

c.生成验证码。SecOC模块应通过传递DataToAuthenticator及DataToAuthenticator的长度等信息，经专门的算法计算后生成授权码MAC。

d.构造安全的I-PDU。SecOC模块应通过将新鲜度值和MAC添加到“真实I-PDU”来构造“安全I-PDU”。

e.增加新鲜度计数器。将新鲜计数器增加1。当然如果在发送前取消传输，则不应增加新鲜度计数器。

f.发送出“安全I-PDU”。

⑤此后，SecOC模块担当上层通信模块的角色，从而为所有较低层请求提供有关“安全I-PDU”数据信息。

⑥最后，将确认信息（是否传输“安全I-PDU”）通知到上层通信模块，以确认是否需要传输“真实I-PDU”。

SecOC模块解耦了上层模块和下层模块之间的交互作用。它能获得所有需要传输的信息，因而能管理与底层模块的相互作用并且不影响上层数模块。

要启动“真实I-PDU”的传输，上层模块应调用PduR_Transmit之类接口。然后，PduR模块将此请求路由到SecOC模块，以便SecOC模块可以立即访问上层通信模块缓冲区中的“真实I-PDU”。

（2）信息“验证”过程

信息“验证”指将“安全I-PDU”中包含的身份验证信息与根据本地数据标识符DataId、本地刷新值Freshness和“真实I-PDU”计算的身份验证信息进行比较的过程。

下面以总线接收来的“安全I-PDU”为例，描述与PduR模块的总体交互及验证“安全I-PDU”的主要步骤。

①对于从较低层总线接口模块传入的“安全I-PDU”的每个请求，SecOC模块都担当上层通信模块的角色，从而服务所有需要的底层请求，以接收完整的“安全I-PDU”。

②SecOC模块将“安全I-PDU”复制到自己的内存中。

③此后，若“安全I-PUD”可用，并在下一次预定调用MainFunction期间，SecOC模块根据要求验证“安全I-PDU”的内容。其详细步骤如下。

a.解析“真实I-PDU”、新鲜度值和授权码MAC。当接收到一个“安全I-PDU”时，SecOC将解析“真实I-PDU”、新鲜度值和它的授权码MAC。

b.构造新鲜度值。SecOC模块应构造新鲜度验证值

Freshness Verify Value（即用于验证的新鲜值）。如果在“安全I-PDU”中传输了完整的新鲜度值，则需要验证构造的Freshness Verify Value是否大于最后存储的新鲜度值。如果它不大于最后一个存储的新鲜度值，SecOC模块应停止验证并丢弃“安全I-PDU”。

c.准备授权码数据。SecOC模块应构造用于计算接收端的授权码数据（DataToAuthenticator）。

DataToAuthenticator=数据标识符DataId|真实I-PDU|完整的新鲜度值

d.验证身份验证信息。SecOC模块应通过DataToAuthenticator、其长度、密钥来验证从“安全I-PDU”中解析出来的验证值。SecOC模块应根据其当前的调用函数SecOC_VerificationStatusCallout和SecOC_VerificationStatus接口，报告每个验证状态（最终状态及各中间状态）。

e.设置新鲜度值。如果对“安全I-PDU”验证成功，则SecOC模块应设置与成功使用的新鲜度值相对应的新的新鲜度值。

f.将“真实I-PDU”传递到上层。只有在验证了“安全I-PDU”成功后，SecOC模块才能使用PduR的下层接口将“真实I-PDU”传递到上层通信模块。如果验证最终失败，则SecOC模块不能将“真实I-PDU”传递给PduR模块。

④如果验证失败，SecOC模块将删除“安全I-PDU”。

⑤如果验证成功，SecOC模块将承担较低层通信模块的作用，并调用PduR_SecOCRxIndication来实现真正的I-PDU。

⑥SecOC模块报告核查结果。

同样，这样做可以分离上层模块和下层模块之间的交互作用。SecOC模块管理与下层模块的交互，直到将完整的“安全I-PDU”复制到自己的缓冲区中。此后，它验证“安全I-PDU”的内容，并依赖于验证结

果，决定是否启动“真实I-PDU”到上层通信模块的传输。

SecOC模块使用了以消息身份验证代码（MAC）为主的对称身份验证方法。它们实现的安全级别与不对称方法的密钥相比要小得多，并且可以在软件和硬件上紧凑、高效地实现。图10.9简单地描述了两个控制器之间SecOC模块的应用示例，以验证基于PDU的ECU在车辆体系内通信的真实性和新鲜性。

图10.9　两个控制器间SecOC模块的应用示例

发送方和接收方都各自集成了SecOC模块，SecOC模块通常与PduR模块交互。为了提供消息的新鲜度值，发送和接收端的SecOC模块为每个唯一可识别的“安全I-PDU”（即每个安全通信链路）维护新鲜度值（如新鲜度计数器、时间戳等）。

在发送方，SecOC模块通过向传出的“真实I-PDU”添加身份验证信息来创建一个“安全I-PDU”。身份验证信息包括一个身份验证器（如消息身份授权代码）和可选的新鲜度值。无论新鲜度值是否包含在“安全I-PDU”有效负载中，都将在验证器生成期间考虑新鲜度值。当使用新鲜度计数器而不是时间戳时，在向接收方提供身份验证信息之前，新鲜度计数器将递增。

在接收方，SecOC模块通过验证发送方SecOC模块追加的身份验证信息来检查“真实I-PDU”的新鲜度和真实性。为了验证“真实I-PDU”的真实性和新鲜性，接收方的“安全I-PDU”应与发送方的“安全I-PDU”完全相同，且接收方SecOC应知道发送方给出的新鲜度值。

10.2 Adaptive AUTOSAR平台

10.2.1 Adaptive AUTOSAR缘起

如前所述，传统的AUTOSAR主要从OSEK演变而来，最开始应用的领域以动力系统方面居多，其操作系统多强调实时性和可靠性，功能相对简单。目前，汽车的架构几乎没什么变化，制动和转向模块必须向节点发送大量消息，这些节点还必须“了解”各个单元及其所处位置。

因此传统AUTOSAR在当前分布式电子电气架构下，第一次基本实现了子系统域内部，尤其是动力子系统内部的功能分布式开发以及集成。

而未来的汽车电子电气架构会更加自由也更具多样性，节点无须了解正在进行通信的计算机处于什么位置，或是将传递什么信息。信息娱乐系统和高级驾驶辅助系统（Advanced Driver Assistant System，ADAS）等电子电气架构势必会对车辆的通信功能提出更高要求，这样才能保证不同模块之间的快速交互。电子控制和通信系统的快速发展，提高了汽车行业对AUTOSAR的接受程度。互联功能、高级安全功能和自动驾驶功能都将改变未来汽车的电子架构，这是因为这些功能采用的控制器之间的通信更加随机，并不像当今绝大多数模块那么固定。

随着汽车电子化程度的逐步深入，整个车辆中还应用了众多其他的操作系统，如QNX、Linux、Android等；另外，像ADAS、机器学习之类的应用对计算量、数据存储等要求特别高，新引入的ARM类芯片，其以太网（Ethernet）通信速度达到惊人的1000M……这些已经不似传统的嵌入式开发，更酷似BAT的开发模式了，由低处理量向高处理量转变，由低通信速率向高通信速率转变，由面向信号流的通信转向面向服务（Service-Oriented）的通信，这些变化都迫切要求传统的AUTOSAR

能够适应兼容这些新的需求。

基于上述技术发展趋势，汽车电子电气架构在将来可能呈现出以下发展趋势，如图10.10所示。

图10.10　汽车电子电气架构发展趋势

正因如此，汽车开放系统架构（AUTOSAR）联盟正推出一套新的标准，以适应更加多变的通信模式。新标准名为自适应AUTOSAR平台（AUTOSAR Adaptive Platform，AP）。该平台经过专门研发，可以为工程师的架构设计提供更大的灵活性。自适应AUTOSAR平台将为复杂性更高的系统提供一个软件框架，帮助工程师利用以太网增加带宽。通过新的自适应平台，AUTOSAR旨在为新的应用提供一个最优的标准化软件框架，特别是对于汽车互联、高度自动化和自动驾驶领域的应用。

AUTOSAR组织在开发自适应AUTOSAR平台时，遵循了以下几个原则：

①能巩固满足现有的需求，如诊断、实时、多核、功能安全、信息安全等；

②能满足新的需求，如新的开发编程模式、文件处理、高性能计算库、在线软件升级等；

③能适应未来变化的需求，如兼容适用当前不同的操作系统、面向服务的通信架构、不同的通信架构等。

基于上述原则，自适应AUTOSAR平台所具备的特色功能如图10.11所示。

图10.11　自适应AUTOSAR平台所具备的特色功能

对于自适应AUTOSAR平台，其所谓的“自适应”（Adaptive）主要表现为原先在经典AUTOSAR平台中设计阶段需静态固化的对象在自适应AUTOSAR上可以动态指定，原先适应性比较小的范围扩展为较大的范围等，例如：

①扩展支持多种不同的操作系统，如Android、Linux等；

②应用程序不但支持静态加载，也支持动态加载；

③通信时信息的提供方不但支持静态指定，还支持实时动态寻找；

④支持类似Linux的开发集成方式；

⑤所有的内存空间对应用程序而言都是虚拟的，可在程序加载时动态分配等。

10.2.2 AP和CP

自适应AUTOSAR平台（AP）并不是传统经典AUTOSAR平台（CP）的替代品，不同的版本可同时存在于同一个车辆中，两个ECU间可通过一些途径，例如以太网，将经典应用和自适应性应用进行无缝衔接。

简单而言，两者的应用场景不太一样：经典AUTOSAR平台多应用于注重硬实时和安全的嵌入式系统中；而自适应AUTOSAR平台则侧重于高性能计算等应用场合，诸如ADAS、互联功能V2X、图像处理、信息娱乐系统等的开发。为此，AP相比于CP做了众多改进，增添了许多新特性，如：

①支持MMU；

②支持OTA“空中”升级功能；

③支持高速数据处理；

④支持类POSIX接口操作系统；

⑤支持渐进部署，即允许动态初始化应用及服务；

⑥支持面向服务的通信架构SOA（Service-Oriented Architecture）；

⑦支持应用程序在RAM中执行；

⑧支持面向对象的C++14开发（复杂算法用C++，安全相关用C）。

按照AUTOSAR组织的发布计划（图10.12），由于自适应

AUTOSAR平台还在开发过程中，其最终的规范1.0预计在2018年10月之后发布，所以最终的发布版本中的特性可能还有增加。按照正常的开发周期，后续各供应商如ETAS、Mentor、Vector大概在AP规范正式发布2年后有对应的产品可用。

图10.12　自适应AUTOSAR平台开发计划

与CP平台相比，AP平台在规范上引入了如虚拟机（Machine）（类似准虚拟ECU，一个真实的ECU可以运行在若干个虚拟机上）、清单文件（Manifest）、功能集群（Clusters）等诸多新概念。

从方法论上讲，AP平台牵涉的开发元素和集成步骤与CP平台有很多的不同。例如：应用软件组件配置信息使用清单文件（Manifest）；集成过程也不是所有的代码经编译后生成.obj后全部链接在一起而不再改变等。可以想象，将来面世的配置工具、静态代码、文件夹结构、编译集成方法等都会与当前的CP大相径庭。已发布的AP方法论规范中定义的自适应AUTOSAR平台相关新元素如图10.13所示。

图10.13　Adaptive AUTOSAR方法论

图10.14中简单地展示了自适应AUTOSAR平台的软件集成流程，给读者一个感性的认识。

图10.14　自适应AUTOSAR平台的软件集成流程

据目前已开发完毕的规范文档来看，Adaptive　AUTOSAR软件架构

大致如图10.15所示。

图10.15　Adaptive AUTOSAR软件架构

直观地看，CP平台中的ASW/RTE/BSW三层架构已不见踪影。整个BSW被划分为自适应平台基础（Adaptive Platform Foundation）和自适应平台服务（Adaptive Platform Services）两大块；另外，原基础软件中的各小模块都不见了，被一个个功能集群（如运行管理集群Execution Management Cluster）所取代；而CP中重要的RTE（Runtime Environment）也被ARA（AUTOSAR Runtime Environment for Adaptive Applications）所代替。

10.2.3　Adaptive AUTOSAR平台新概念介绍

下面就目前Adaptive AUTOSAR平台所提出的几个新概念和新特性进行简要介绍。

（1）AUTOSAR自适应应用

AUTOSAR自适应应用（Adaptive Application，AA）类似于CP中的软件组件SWC（图10.16），其采用面向对象的C++开发，可在不同的状态下运行不同的线程（Thread）。其接口主要有以下三类：

图10.16 AUTOSAR自适应应用

①与POSIX OS之间的PSE51接口；

②与Services之间的ara：：com接口；

③与Foundation之间的API。

（2）AUTOSAR自适应平台基础

简单地理解，自适应平台基础（Adaptive Platform Foundation，FO）就是把CP和AP两平台中通用的需求和技术规范抽取出来组成了Foundation，其目的是强化两平台协同工作的能力，如图10.17所示。

图10.17　AUTOSAR自适应平台基础

自适应平台基础和自适应平台服务两大块中都包含众多的功能集群（Functional Cluster），如通信集群、诊断集群等，当前发布的版本中的功能集群如表10.1所示。

表10.1　功能集群

功能集群 Functional Cluster	简称 Short Name	主要功能 Main Functions
通信管理(Communication Management)	com	AA 间的通信、服务发现 SD、中间件通信
运行管理(Execution Management)	exec	虚拟机的启停、虚拟机的状态管理、 AA 的启停、权限管理和控制
操作系统(Operating System)	os	
诊断(Diagnostics)	diag	
固存管理(Persistency)	per	非易失性数据读写、安全存储
记录和追踪(Log and Trace)	log	信息登记和跟踪等
平台健康管理(Platform Health Management)	phm	程序流监控、程序存活监控等
表述性状态转移服务(REST)	rest	
时间同步(Time Synchronization)	time	提供各种全局或本地时基
Id 访问管理(Identity Access Management)	iam	身份认证、服务授权
加密管理(Cryptography)	crypto	加密/解密
信号转服务(Signal to Service)	s2s	把诸如 CAN 信号转为各种服务接口
更新和配置管理(Update & Configuration Management)	ucm	安装、更新、卸载软件、签名验证、软件回滚

（3）AUTOSAR自适应应用运行接口

AUTOSAR自适应应用运行接口（AUTOSAR Runtime Environment for Adaptive Applications，ARA）类似于RTE。它又可分为两类形式：一种是与FO之间的直接接口，表示为各种API；另一种是各种Services之间的通信接口，表示为ara：：com（图10.18）。和CP的RTE做个类比的话，可以把ara：：com看作Rte_Write、Rte_Read、Rte_Send、Rte_Receive、Rte_Call、Rte_Result等RTE APIs。

图10.18　AUTOSAR自适应应用运行接口

自适应应用可以经过ara：：com接口实时建立连接后通信，该过程需要通过服务发现模块（Service Discovers，SD）。

（4）虚拟地址空间

经典AUTOSAR平台的程序代码通常运行在ROM中，可视不同的ASIL等级把ROM空间进行隔离（Memory Partition），但无论如何都是在设计阶段事先静态分配的，即无论是ROM还是RAM对程序而言在运行之前存储空间都是确定的，且所有的Applications都共享一个地址空间。而Adaptive AUTOSAR则支持直接虚拟化地址空间（Address Space Virtualization），给每个应用进程分配独立的地址空间。地址空间可以经内存管理单元（Memory Management Unit，MMU）保护，以防非授权的恶意访问，以求更好地满足功能安全的要求。对应用程序而言实际的地址空间可能是不固定的，只有在程序加载时才最终确定，另外实际的物理地址对应用程序而言也是不可见的。

（5）固存集群

在Adaptive AUTOSAR中，固存集群（Persistence Cluster）给自适

应应用和其他平台化的功能集群提供基于数据库访问的数据服务，其包括：

①在程序启动（BOOT）和上下电循环过程中保存数据；

②使用唯一的Id访问数据；

③类文件系统的数据读/写；

④固存数据的加密；

⑤保存数据的错误检测和校正。

固存集群在AP平台中起到类似于CP平台中存储协议栈（Memory Stack）的功能。但差别在于其存储主要是以键-值对（Key-Value）和文件流（Stream）的形式，如图10.19所示。

图10.19　固存集群存储过程示意

（6）POSIX操作系统

POSIX（Portable Operating System Interface，可移植操作系统接口）是一个应用和操作系统间的标准编程接口，虽然其并非发源于汽车工业，但POSIX操作系统让汽车电子软件开发变得更加便捷和灵活。POSIX有众多不同的子集，Adaptive AUTOSAR Platform OS部分向其他部分提供了最小的接口规范子集PSE51，以便更好地在不同的操作系统和应用间做移植（图10.20）。POSIX接口让用户可以集中精力于应用开发，且可任意使用于不同的控制器。

图10.20　POSIX可移植操作系统接口

（7）面向服务的通信

传统的CP平台多采用面向信号的通信（Signal-Oriented Communication）方式。面向信号的通信，顾名思义，这种通信主要以信号为中心，关注信号流的走向，将信号从一个点发送至另一个点。假设两个独立的ECU，比如变速箱控制器（Transmission Control Unit，TCU）需要混合动力控制单元（Hybrid Control Unit，HCU）的某一个信息（如期望的扭矩输出），HCU就直接把这个信号同其他信号一起，打包成报文发到总线上，典型的如CAN总线。TCU收到之后解包就能够获得该信号。在这种传输情况下最核心的就是通信矩阵，如DBC文件中所定义的：信号本身的属性，如信号名、信号位宽、帧Id、发送周期等，以及发送/接收的节点等。

在介绍面向服务的通信（Service-Oriented Communication，SOC）概念之前，先介绍一下面向服务的体系结构（Service-Oriented Architecture，SOA）。SOA是一种基于请求/应答设计范式的分

布式计算的进化，用于同步和异步应用程序。应用程序的业务逻辑或单个函数模块化，并作为消费者/客户端（Consumer/Client）应用程序的服务呈现。这些服务的关键是它们松散耦合的性质；也就是说，服务接口独立于实现。应用程序开发人员或系统集成商可以通过撰写一个或多个服务而不知道服务的底层实现来构建应用程序。

面向服务的通信架构SOA中的服务注册中心是一个服务和数据描述的存储库。服务提供者通过注册中心发布它们的服务，而服务使用者可以向注册中心发现和查找可用的服务，因为服务（Service）部署在服务器端（Server），在具体实现（实例化）时会有些参数，例如网络地址，可能会发生变化。为了能够让客户端随时找到服务器上的服务，因此需要使用服务发现（Service Discovery，SD）机制。上述过程的示意如图10.21所示。

图10.21　面向服务的通信机制示意

在SOME/IP（Scalable service-Oriented MiddlewarE over IP）SD中包含了发现服务、提供服务、停止提供服务、订阅事件组、停止订阅事件

组等7种类型的服务发现相关报文，SD模块实现了服务发现的内容、管理服务的地址信息、状态信息等内容；而SOME/IP模块实现了数据的传输控制，用于服务接口数据的上下传递。

服务提供者和服务使用者直接的沟通路径通常可在设计阶段建立，也可以在实时运行时动态建立。如图10.22所示，系统中有三个服务实例（Application 2、Application 3和Application n），这些都被Application 1中的ServiceFind（）找到了，然后会被代理标记为P1、P2和P3。Application 1可以任选一个作为服务的提供者完成后续的服务。当然Proxy与Skeleton之间是通过中间件（Middleware）完成最终通信的。

图10.22　面向服务的通信实例

遵循AP规范开发的应用程序拥有一个独特的优势：它们可以在运行时安装进系统中，犹如智能手机中的各种APP。这意味着面向服务的通信可让APP单独地开发、测试、更新或发布，并在任意时刻集成进整个系统中。所有AP中的软件组件都会使用面向服务的通信（SOC），这些通信的建立可以在设计阶段确定好，也可在实时运行时动态建立，这点与CP相比是一个巨大的不同。

总结一下，AP和CP两个平台主要的差异如表10.2所示。

表10.2　AUTOSAR经典平台与自适应平台对比

AUTOSAR 经典平台(CP)	AUTOSAR 自适应平台(AP)
C 语言开发	C++语言开发
基于 OSEK	基于 POSIX(PSE51)
从 ROM 中执行	应用程序可载于 RAM 中运行
所有 Applications 共享一个地址空间	每个 Applications 都有一个独立的地址空间
关注于面向信号的通信(CAN、FlexRay 等)	关注于面向服务的通信(SOME/IP)
任务静态配置	支持动态调度策略
文档详细描述各模块	规范轻模块，重模块描述
各协议栈一起编译和集成	软件组件为可加载的 POSIX 进程，可分开编译加载

10.3　本章小结

随着新技术的不断应用，以及车联网领域内对车辆信息安全的持续关注，AUTOSAR规范标准也相应对这两个领域进行了拓展。本章添加了AUTOSAR与信息安全以及自适应AUTOSAR平台相关的普及性知识。从而引导读者对AUTOSAR规范未来发展趋势有一定的了解。

AUTOSAR　CP和AP将在汽车电子行业内逐步取代旧式的ECU软件架构，与其规范化的方法论一起被越来越多的厂家所学习、讨论和接受。

参考文献

［1］ 李建秋，赵六奇，韩晓东等.汽车电子学教程［M］.第2版.北京：清华大学出版社，2011.

［2］ AUTOSAR GbR.Layered Software Architecture［EB/OL］.（2017-12-08）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/43/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

［3］ AUTOSAR GbR.Methodology［EB/OL］.（2017-12-08）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/43/AUTOSAR_TR_Methodology.pdf.

［4］ 单忠伟.符合AUTOSAR规范的燃料电池汽车动力系统能量管理控制模块开发［D］.上海：同济大学，2017.

［5］ Franco F R，Neme J H，Santos M M，et al.Workflow and toolchain for developing the automotive software accordir ECU［C］.IEEE International Symposium on Industrial Electronics，2016：869-875.

［6］ AUTOSAR GbR.Specification of Communication［EB/OL］.（2011-11-16）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/40/AUTOSAR_SWS_COM.pdf.

［7］ AUTOSAR GbR.Specification of PDU Router［EB/OL］.（2010-10-

26）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_PDURouter.pdf.

［8］ AUTOSAR GbR.Specification of CAN Interface［EB/OL］.（2011-12-
01）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_CANInterface.pdf.

［9］
AUTOSAR GbR.Specification of Communication Manager［EB/OL］.（2011-12-
05）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_COMManager.pdf.

［10］ AUTOSAR GbR.Specification of CAN State Manager［EB/OL］.（2011-11-
24）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_CANStateManager.pdf.

［11］ AUTOSAR GbR.Specification of ECU State Manager［EB/OL］.（2011-11-
24）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_ECUStateManager.pdf.

［12］
AUTOSAR GbR.Specification of Basic Software Mode Manager［EB/OL］.（2011-12-
09）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_BSWModeManager.pdf.

［13］ ETAS.RTA-RTE V6.0.0 Reference Manual.2016.

［14］ AUTOSAR GbR.Specification of RTE［EB/OL］.（2011-10-

26）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_RTE.pdf.

［15］ ETAS.RTA-OS Reference Guide.2016.

［16］ AUTOSAR GbR.Specification of Operating System［EB/OL］.（2011-11-
23）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_OS.pdf.

［17］ 陈海兰，罗晓敏，涂时亮等.基于AUTOSAR的实时操作系统设计与实现［J］.计算机工程，2012，38（20）：9-12.

［18］ NXP Semiconductors.MPC5744P Reference Manual.2015.

［19］ AUTOSAR GbR.Specification of MCU Driver［EB/OL］.（2012-12-
09）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_MCUDriver.pdf.

［20］ AUTOSAR GbR.Specification of GPT Driver［EB/OL］.（2011-10-
13）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_GPTDriver.pdf.

［21］ AUTOSAR GbR.Specification of PORT Driver［EB/OL］.（2010-11-
03）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_PortDriver.pdf.

［22］ AUTOSAR GbR.Specification of DIO Driver［EB/OL］.（2011-09-
30）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4
0/AUTOSAR_SWS_DIODriver.pdf.

［23］ AUTOSAR GbR.Specification of ADC Driver［EB/OL］.（2011-11-
24）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4-0/AUTOSAR_SWS_ADCDriver.pdf.

［24］ AUTOSAR GbR.Specification of PWM Driver［EB/OL］.（2011-10-
04）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4-0/AUTOSAR_SWS_PWMDriver.pdf.

［25］ AUTOSAR GbR.Specification of ICU Driver［EB/OL］.（2011-11-
02）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4-0/AUTOSAR_SWS_ICUDriver.pdf.

［26］ AUTOSAR GbR.Specification of CAN Driver［EB/OL］.（2011-11-
02）.https：//www.AUTOSAR.org/fileadmin/user_upload/standards/classic/4-0/AUTOSAR_SWS_CANDriver.pdf.

［27］ ISO26262-4.Road vehicles-Functional safety-Part 4：Product development at the system level.ISO，2011.06.27.

［28］ ISO26262-5.Road vehicles-Functional safety-Part 5：Product development at the hardware level.ISO，2011.06.27.

［29］ ISO26262-6.Road vehicles-Functional safety-Part 6：Product development at the software level.ISO，2011.06.27.

［30］ AUTOSAR GbR.Specification of Secure Onboard Communication［EB/OL］（2017-12-
08）.https：//www.autosar.org/fileadmin/user_upload/standards/classic/4-

3/AUTOSAR_SWS_SecureOnboardCommunication.pdf.

［31］
AUTOSAR GbR.Specification of Crypto Service Manager［EB/OL］.（2017-12-
08）.https：//www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_CryptoServiceManager.pdf.

［32］ AUTOSAR GbR.Specification of SW-C End-to-End Communication Protection Library［EB/OL］.（2017-12-08）.https：//www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_E2ELibrary.pdf.

［33］
AUTOSAR GbR.Explanation of Adaptive Platform Design［EB/OL］.（2017-10-
27）.https：//www.autosar.org/fileadmin/user_upload/standards/adaptive/17-10/AUTOSAR_EXP_PlatformDesign.pdf.