面向FreeRTOS的AUTOSAR RTE设计与实现

栗花伟[1,2]　黄晶晶[2]　张昱[1]　许东辉[3]　孙涛[3]

（1.中国科学技术大学，合肥 230031；2.合肥综合性国家科学中心人工智能研究院，合肥 230088；3.安徽江淮汽车集团股份有限公司，合肥 230022）

【摘要】针对汽车开放系统架构（AUTOSAR）工具链对商业操作系统依赖性高且对免费实时操作系统（FreeRTOS）适配不足的问题，提出了一种基于模块化架构的运行时环境（RTE）代码自动生成工具。该工具通过信息提取器、解析器、处理模板和代码生成器等模块，实现了FreeRTOS环境下符合AUTOSAR规范的RTE代码自动生成。测试结果表明，所生成代码在结构规范性、类型安全性和实时性能等方面满足工业级要求，单模块开发时间由8.2人时降至0.5人时，跨平台适配效率提升12.7倍。研究成果为智能汽车嵌入式软件开发提供了高效可靠的解决方案。

主题词：AUTOSAR规范　运行时环境　FreeRTOS适配　代码自动生成　模块化设计

中图分类号：TP316.8；U469.72　　**文献标志码：**A　　**DOI:** 10.19620/j.cnki.1000-3703.20250304

Design and Implementation of an AUTOSAR RTE for FreeRTOS

Li Huawei[1,2], Huang Jingjing[2], Zhang Yu[1], Xu Donghui[3], Sun Tao[3]

（1. University of Science and Technology of China, Hefei 230031; 2. Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei 230088; 3. Anhui Jianghuai Automobile Group Co., Ltd., Hefei 230022）

【Abstract】To address the high dependency of AUTOSAR toolchains on commercial operating systems and insufficient FreeRTOS adaptation, a modular RTE code auto-generation tool is proposed. The tool integrates components such as an information extractor, parser, processing template, and code generator to automatically produce RTE code compliant with AUTOSAR standards in FreeRTOS environments. Test results show that the generated code meets industrial requirements in structural compliance, type safety, and real-time performance. Development time per module is reduced from 8.2 man-hours to 0.5 man-hours. Cross-platform adaptation efficiency increases 12.7 times. This research provides an efficient and reliable solution for embedded software development in intelligent vehicles.

Key words: AUTOSAR Standard, Runtime environment, FreeRTOS adaptation, Code auto-generation, Modular design

【引用格式】 栗花伟, 黄晶晶, 张昱, 等. 面向FreeRTOS的AUTOSAR RTE设计与实现[J]. 汽车技术, 2026(7): 36-44.

LI H W, HUANG J J, ZHANG U, et al. Design and Implementation of an AUTOSAR RTE for FreeRTOS[J]. Automobile Technology, 2026(7): 36-44.

1　前言

随着汽车电子系统向智能化、电动化演进，电子控制单元（Electronic Control Unit，ECU）软件架构日益复杂，安全需求急剧上升[1]。智能汽车代码量已突破1亿行，传统人工编码与模块化开发模式面临耦合高、迭代慢等挑战[2-3]。在此背景下，汽车开放系统架构经典平台（Automotive Open System Architecture Classic Platform，AUTOSAR CP）通过分层架构实现软硬件解耦，其核心模块运行时环境（Runtime Environment，RTE）在应用层软件（Application Software，ASW）与基础软件层（Basic Software，BSW）之间提供标准化接口，保障数据传输、服务调用与任务调度的高效性[4-6]。传统AUTOSAR生态中，车载电子开放系统及其接口操作系统（Offenbach Systeme Elektronik Karlsruhe Operating System，OSEK OS）因其静态优先级调度机制，长期占据

通信作者：张昱（1972—），教授，主要研究方向为面向新兴计算的编程系统、智能无人系统、软件分析与系统优化，yuzhang@ustc.edu.cn。

车载实时操作系统主导地位[7]。

然而，智能汽车硬件平台日益多样化，对底层操作系统（Operating System，OS）提出新要求。OSEK OS存在以下问题：一是需为不同平台定制版本，导致开发成本高、复用性差；二是闭源特性在供应链波动下制约了国产化进程[8]。相比之下，FreeRTOS具备轻量内核、动态任务支持及多架构兼容优势，成为OSEK OS的替代技术优选[9]。但由于两者在任务调度与资源管理机制上差异显著，直接替换会造成大量软件组件（Software Component，SWC）与BSW模块适配困难，增加适配复杂性与出错风险。

适配难点集中在RTE层的接口兼容性。RTE作为AUTOSAR架构的“通信中枢”，需准确对接操作系统服务与上层应用。但FreeRTOS缺乏原生AUTOSAR接口，导致需手动重构通信逻辑与任务映射，适配周期延长3~5倍，且易引入时序错误（如优先级反转、资源竞争）。

目前，FreeRTOS与AUTOSAR的兼容性研究尚处于初级阶段，缺乏成熟工具链支持。国际主流工具链（如ETAS、Vector）尚未开放FreeRTOS支持套件，国内基础软件生态仍属空白。国外研究集中在AUTOSAR架构优化[10-12]，如Khenfri等[10]提出的实体任务映射算法多基于OSEK OS；国内研究多聚焦AUTOSAR应用层设计[13-18]，如郑燚等[16]的以太网通信系统验证工作，但针对FreeRTOS的高效代码生成工具仍待深入探索。

针对以上问题，本文拟设计并研发一款适配FreeRTOS的RTE代码生成工具，对以下方面开展研究：

a. 构建标准适配层，旨在解决SWC与BSW之间的接口兼容问题。

b. 探索并打通AUTOSAR CP与FreeRTOS的底层调度机制。

c. 研究自动代码生成方法，从而减少人工适配工作量并降低潜在错误风险。

本研究计划在NXP S32K3平台上对该工具的OS迁移效率进行工程化验证，以期为智能汽车软件的国产替代探索可行路径。

2 RTE生成器总体架构设计

为解决FreeRTOS与AUTOSAR系统的兼容性问题，并克服手动适配方法的低效性，本文提出一种名为XBus消息总线生成器的RTE代码自动生成工具，旨在提供高效、自动化的RTE代码生成方案，实现FreeRTOS兼容性优化，并提升开发效率和系统灵活性。该生成器采用模块化设计，包括信息提取器、XBus解析器、XBus处理逻辑模板、OS配置器及XBus代码生成器，其架构如图1所示。模块的功能如下：

a. 信息提取器。从AUTOSAR ECU描述文件中提取生成RTE目标代码所需的信息，并输出XBus消息总线描述文件（JSON格式）。

b. XBus解析器。解析XBus消息总线描述文件，提取必要的配置信息并生成适配XBus处理逻辑模板文件的信息数据，同时为OS配置器提供可运行实体信息。

c. XBus处理逻辑模板。提供符合RTE标准的逻辑框架代码，作为代码生成的模板。

d. OS配置器。支持用户交互配置任务信息，完成可运行实体到任务的映射，并将任务配置信息返回给XBus解析器。

e. XBus代码生成器。根据XBus解析器生成的完整信息数据和处理逻辑模板文件，生成目标RTE代码。

XBus消息总线生成器的流程如图2所示，主要包括以下步骤：

a. 用户自定义输入和输出路径，输入为ECU所有的ARXML（AUTOSAR可扩展标记语言文件，AUTOSAR XML）描述文件，输出为RTE层的源码。

b. 通过信息提取器提取ARXML文件并生成消息总线描述文件。

c. XBus解析器解析该描述文件，生成初步的信息数据（包含SWC、BSW的信息，如可运行实体、通信API等），并将可运行实体相关的信息数据传递给OS配置器供用户进行任务配置和映射。

d. 配置完成后，XBus解析器整合任务配置信息生成完整的信息数据。

e. 由XBus代码生成器结合模板文件生成目标RTE代码，并将目标代码输出到指定路径。

通过上述设计与实现，本文提出的RTE生成器方案不仅能够有效解决FreeRTOS与AUTOSAR系统之间的兼容性问题，还为汽车电子系统的开发提供了一种高效、灵活的解决方案。

3 详细设计

3.1 信息提取器设计

3.1.1 符号与辅助函数定义

本节在介绍信息提取器的设计前，首先定义主要符号及辅助函数，如表1所示。

3.1.2 总体设计与功能

信息提取器在XBus架构中起着关键的数据提取与

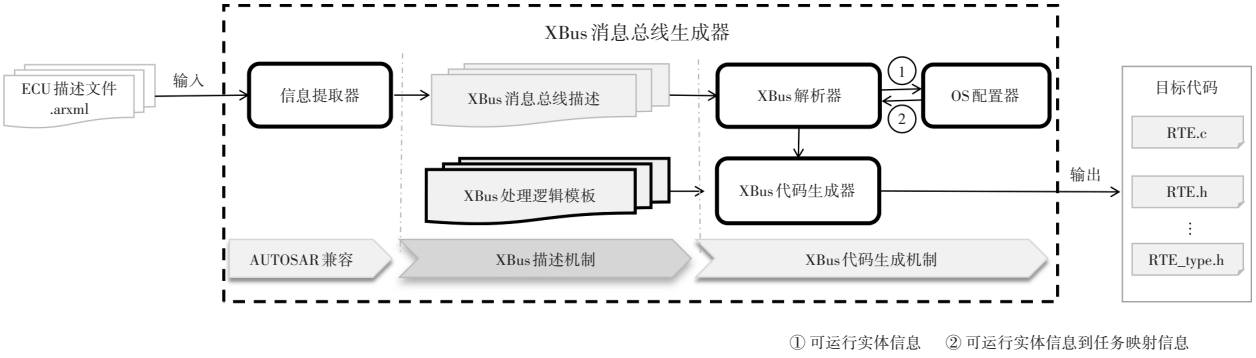


图1 XBus消息总线生成器架构

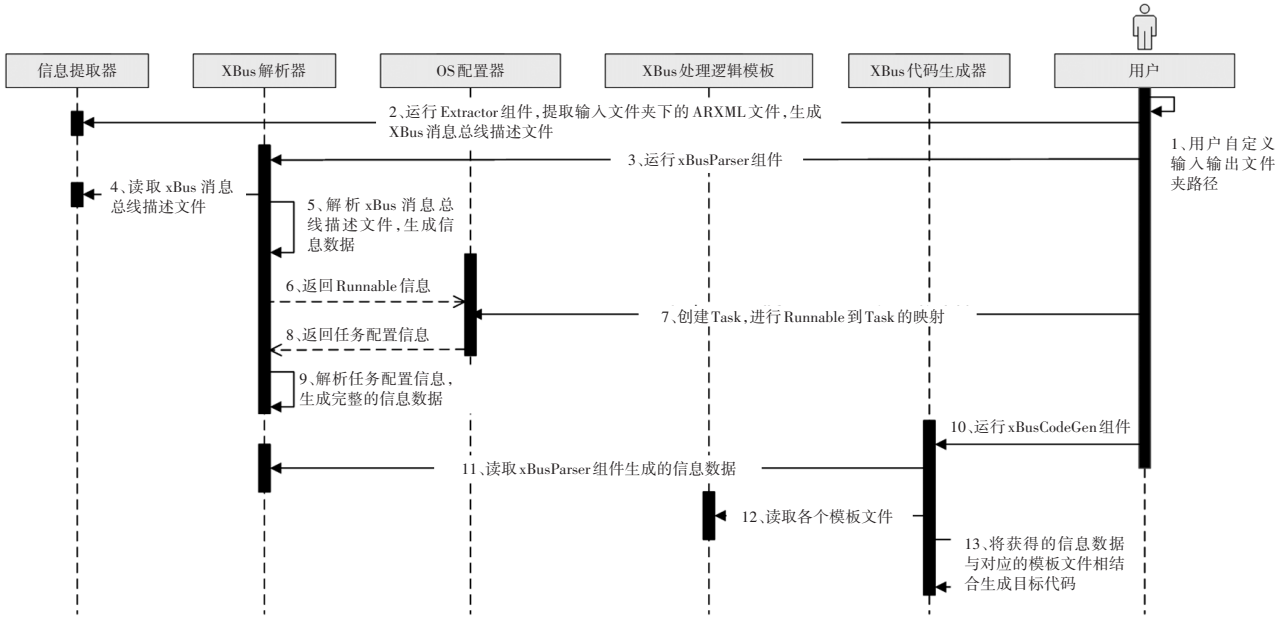


图2 XBus消息总线生成器时序图

表1 符号定义对照表

符号	描述
D_{ARXML}	输入ARXML文档集合
$\{J_{\mathrm{JSON}}\}$	提取器输出的标准化JSON结构
X	AUTOSAR元模型约束的XPath集合，其中XPath（XML Path Language）为用于定位和提取XML文档中特定元素和属性的查询语言
$\Psi \subseteq X$	元数据提取规则集，$\Psi = \{\psi_i\}_{i=1}^{n}$
R	ARXML交叉引用路径集合
$P:(D,\Psi) \rightarrow J$	信息提取过程（$D \in D_{\mathrm{ARXML}}$，$J \in \{J_{\mathrm{JSON}}\}$）
$f_{\mathrm{ref}}: R \rightarrow X$	交叉引用解析
V	DOM树中的节点集合
v	当前解析节点，$v \in V$
θ	递归解析的最大深度阈值
f_{conv}	节点转换函数，将XML节点转换为JSON对象
s	SHORT-NAME标签的值
$\Gamma(s)$	以s为键的JSON对象，存储对象类型和内容
$\oplus$	字符串连接操作符
$count(t)$	当前键t在同一层级中的出现次数

预处理作用，其核心任务是从复杂的ECU描述文件中精准提取与RTE生成紧密相关的信息，并将其转化为适用于后续处理的格式。图3展示了信息提取器的总体结构，其包含以下3个核心子模块：

a. 配置文件。定义基于XPath的元数据定位规则集Ψ。

b. 主解析器。基于lxml库构建的迭代式数据对象模型（Data Object Model，DOM）遍历器，形式化为$P:(D,\Psi)\rightarrow J$，采用递归下降解析算法（详见2.1.3节）提取软件组件、通信端点与事件等RTE参数。

c. 动作依赖解析器。处理ARXML中的交叉引用问题，构建引用解析函数$f_{\text{ref}}:R\rightarrow X$，借助XPath反向索引机制（即预先建立从被引用对象到引用它的所有节点的映射关系）实现从被引用对象到引用节点的映射，提升解析效率并避免全局遍历。

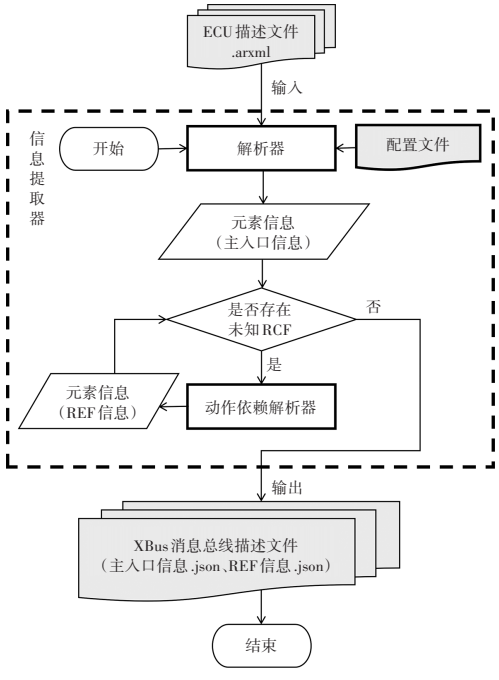


图3　信息提取器结构

3.1.3　解析规则与转换算法

为实现ARXML到JSON的无损转换，本文设计了一套兼顾效率与可扩展性的解析规则体系，其核心算法如图4所示。

a. 递归式节点遍历。对任意节点$v(v\in V)$，采用递归下降方式解析，当节点深度超过阈值θ时终止递归（第2~4行）：

$$f_{\text{conv}}(v)=\begin{cases}\bigcup_{c\in children}f_{\text{conv}}(c), & depth(v)<\theta\\ \{\langle tag(v),text(v)\rangle\}, & depth(v)\geqslant\theta\end{cases}\quad(1)$$

式中：$children$为节点v的所有子节点集合，c为其中的子节点，$depth(v)$为节点v的当前深度，$tag(v)$与$text(v)$分别为节点v的标签名与对应的文本内容。

b. SHORT-NAME映射。利用节点SHORT-NAME属性的全局唯一性，将该属性值作为JSON对象的主键（第8、18行）：

$$\Gamma(s)=\{"obj_type":tag(parent(s)),"content":\Gamma'(s)\}\quad(2)$$

式中：$\Gamma'(s)$为s关联的具体对象内容，该机制将查询复杂度从$O(n)$降至$O(1)$。

c. 键冲突消解。对同层重复标签t，注入后缀@$count$(t)以确保唯一性（第11~13行）：

$$f_{conv}(t)=\begin{cases}t, & count(t)=1\\ t\oplus"@"\oplus count(t), & count(t)>1\end{cases}\quad(3)$$

式中：$\oplus$为字符串连接操作符，$count(t)$为当前标签键t在同一层级中的出现次数。

该提取器通过三重机制实现高效转换：

a. 规则驱动的XPath体系兼容多版本标准。

b. 递归优化实现语义无损转换，查询效率达$O(1)$。

c. 基于反向索引机制有效解析交叉引用，整体提升RTE参数提取效率与准确性。

算法1. ARXML-JSON递归转换算法

输入：当前节点v，当前深度d，最大深度θ
输出：转换后的JSON对象J

```
1: function CONVERT(v, d)
2:     if d ⩾ θ or v is leaf then
3:         return {"value":text(v)}    //规则1：递归终止
4:     end if
5:     J ← {}, s ← null//初始化容器
6:     for each child c of v do
7:         if tag(c) = "SHORT-NAME" then
8:             s ← text(c)//规则2：捕获主键
9:         else
10:            key ← tag(c)
11:            if key ∈ J then
12:                key ← key + "@" + count(J[key])//规则3：键冲突消解
13:            end if
14:            J[key] ← CONVERT(c, d + 1)//规则1：递归下降
15:    end if
16:    end for
17:    if s ≠ null then
18:        return {s : J}//规则2：主键封装
19:    end if
20:    return J
21: end function
```

图4　ARXML-JSON递归转换算法

3.2　XBus处理逻辑模板设计

基于AUTOSAR方法论，本文设计了面向FreeRTOS的XBus处理逻辑模板架构，模板采用Mustache语言实现，具备无逻辑、轻量级等特性，有助于提升模板维护性和跨平台一致性[20]。模板体系由14个文件组成，覆盖任务管理、RTE主函数、组件接口、通信管理等模块（见图5）：

a. 任务调度管理模块。定义FreeRTOS任务创建、调度与优先级。

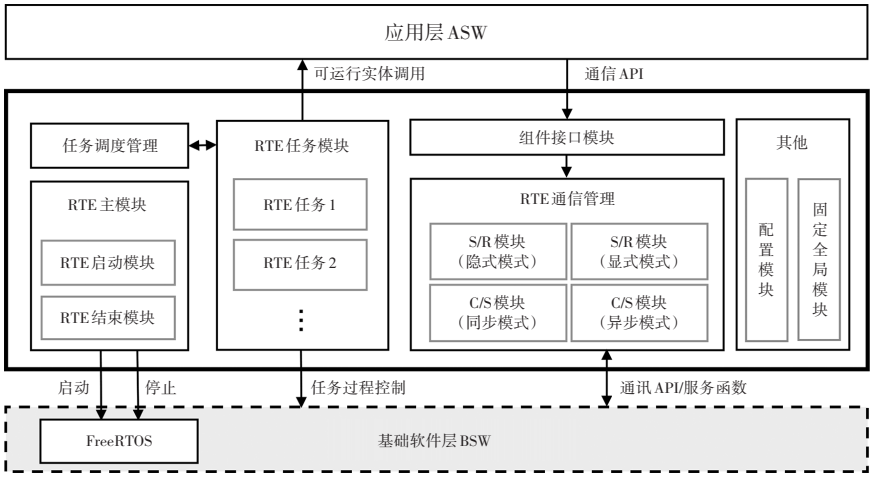


图5　XBus处理逻辑模板架构

b. 组件接口模块。提供ASW与RTE之间的API及数据类型定义。

c. RTE通信管理模块。支持发送者/接收者接口（Sender/Receiver，S/R）和客户端/服务器接口，Client/Server，C/S）通信模式。

d. RTE主模块。实现初始化与关闭操作，保障系统稳定。

e. 配置模块。支持RTE参数化配置。

f. 全局定义模块。统一关键宏和数据结构，确保一致性。

上述模块协同工作，支持自动生成符合FreeRTOS调度与通信机制的RTE代码。以下重点介绍OS配置模板和通信接口模板。

3.2.1　FreeRTOS OS配置模板设计

任务管理模块用于生成FreeRTOS任务配置代码，包括任务优先级、栈大小、周期等参数。图6展示了使用Mustache语法遍历os_tasks_data，自动生成任务栈、控制块与定义结构。其中{{OS_Task}}类占位符将在渲染阶段替换为具体任务参数，确保代码生成灵活性与一致性。

代码1. OS配置模板示例

```
1: {{#os_tasks_data}}
2: static uint32 Os{{OS_Task}}TaskStack[{{OS_Task}}_STK_SIZE];
3: static osStaticThreadDef_t Os{{OS_Task}}TaskTCB;
4: static osThreadDef_t Os{{OS_Task}}TaskDef = {
5:     Os{{OS_Task}},
6:     {{OS_Task}}_PRIO,
7:     1,
8:     {{OS_Task}}_STK_SIZE,
9:     Os{{OS_Task}}TaskStack,
10:    &Os{{OS_Task}}TaskTCB
11: };
12: {{/os_tasks_data}}
```

图6　OS配置模板示例

此外，为适配FreeRTOS，其他模板文件（如rte_c.xtl、rte_cfg_header.xtl等）也做了相应优化，以充分利用其实时调度与内存管理机制。

3.2.2　通信接口模板设计

典型接口包括S/R和C/S。如图7所示代码2，以S/R数据读取为例，模板根据连接类型生成对应逻辑：使用组装连接器时，直接返回组件内部数据；使用委托连接器时，调用底层通信服务获取数据，并通过原子操作确保数据一致性，处理错误状态，增强通信可靠性。

代码2. SR接口通用读取模板伪代码

```
1:  // SR接口读取函数模板
2:  FUNCTION Rte_Read_Data(component, interface)
3:    IF 使用组装连接器:
4:      RETURN component.internal_communication_data
5:    ELSE:
6:      status = Com_ReceiveSignal(signal_id)
7:      IF status == SUCCESS:
8:        ATOMIC_UPDATE(component.data, received_data)
9:      ELSE:
10:       SET read_success = False
11:     RETURN operation_status
12: END FUNCTION
```

图7　SR接口通用读取模板伪代码

3.3　XBus解析器设计

XBus解析器用于解析XBus消息总线描述文件，提取所需的关键信息并构建代码元数据，为OS配置器和代码生成器提供支持。图8所示为XBus解析器结构。

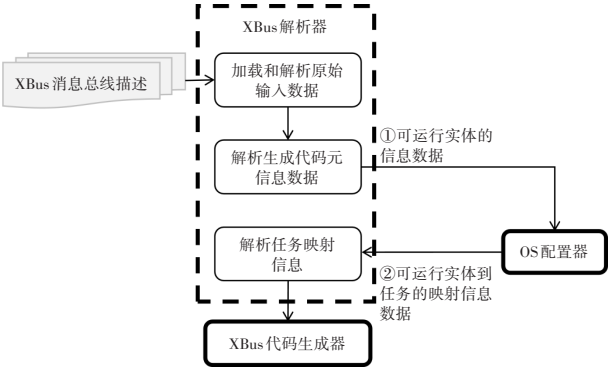


图8　XBus解析器结构

a. 加载和解析原始输入数据。包括解析COM信号、接口映射关系、SWC信息（事件、可运行实体、RTE API）、BSW信息（事件、SchM API）和数据类型。

b. 解析生成代码元信息数据。提取内部连接、外部信号映射、回调函数、与任务解耦的API、未连接接口等信息，并传递可运行实体数据至OS配置器。

c. 解析任务映射信息。从OS配置器获取可运行实体与任务的映射，补全最终元数据，为代码生成器提供输入。

3.4 OS配置器设计

如图9所示，OS配置器依据AUTOSAR分层方法构建任务管理框架，采用三阶段机制实现可运行实体到任务的映射：

a. 交互式任务创建。通过GUI配置任务参数（优先级、堆栈、周期）、触发方式及资源分配（互斥锁、信号量等）。

b. 实体-任务动态映射。通过GUI界面结合XBus解析数据实现可运行实体到任务的可视化绑定，任务负载≤8实体，自动注入SchM_Enter/Exit保障临界区安全，并进行边界检查与反馈。

c. 配置数据回传。将规范化JSON映射数据反馈至XBus解析器，用于生成最终元数据。

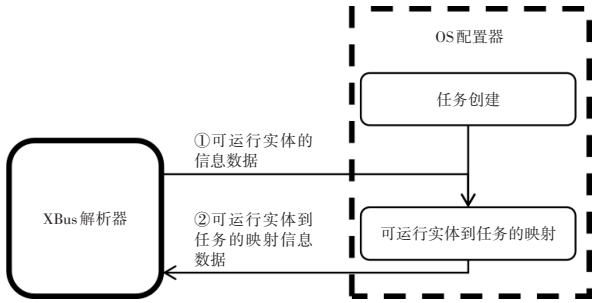


图9 OS配置器结构

3.5 XBus代码生成器架构设计

XBus代码生成器作为领域特定语言到目标代码的转换枢纽，通过元数据驱动模板渲染机制实现模型到代码的自动化转换。如图10所示，系统核心处理流程可形式化为三元组$\langle C,T,M\rangle$，其中C为配置规约，T为模板语法树集合，M为元数据模型。系统包含以下关键阶段：

a. 配置规约范式。使用YAML非标记语言（YAML Ain't Markup Language）配置语言定义代码生成约束，形式化为：

$$C=\{\phi_n,\tau_m,\omega_k\} \tag{4}$$

式中：ϕ_n为文件命名策略，τ_m为模板绑定规则，ω_k为元数据映射约束，规约通过Schema校验确保语法和语义一致性。

b. 模板预处理阶段。通过Mustache模板引擎，实现AST的惰性加载。该过程包括模板解析与语法校验，处理函数表示为：

$$L(t)=\bigcup_{t_i\in T} ParseTemplate(t_i)\circ Validate(\tau_i) \tag{5}$$

式中：$L(t)$为预处理后生成的AST集合；$ParseTemplate()$为模板解析函数；$Validate()$为语法校验函数；t_i为模板集合T中的单个模板实例；$\circ$为函数组合操作，通过静态分析保证模板变量与ω_k约束保持兼容。

c. 多态代码生成。基于上下文感知的变量绑定，执行插值、条件分支与迭代控制，最终依据命名策略生成多层级代码，并通过MISRA-C规则进行预验证，保障生成代码的完整性与合规性。

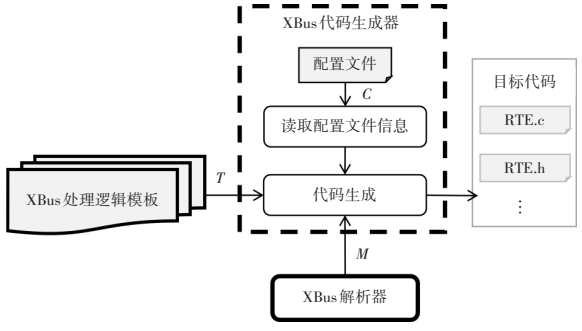


图10 XBus代码生成器结构

如图11所示，算法的输入包含配置规约C_{cfg}、模板集合T_{tpl}和元数据模型M_{meta}，输出为生成文件集合F_{out}。首先初始化持久化结果集合F并语义解析C_{cfg}生成中间表示C_{ir}（算法第1~2行）；再加载模板T_{tpl}为AST形式T_{ast}（第3行）；对T_{ast}中的每个模板实例tpl_i逐一进行类型检查，确保与配置中的元数据映射约束ω_k兼容（第4~8行）；若符合要求，则构建上下文CTX，合并元数据模型M_{meta}和中间表示中的命名策略ϕ_n，并进行部分求值绑定变量，生成求值后的模板实例tpl_{pe}（第9~10行）；随后基于tpl_{pe}生成中间代码code，并依据命名策略计算出对应的文件名fname。若存在同名文件且配置中的覆盖标志位$C_{ir}.\tau_m.overwrite$为假（即不允许覆盖），则使用代码的SHA1值做版本管理以更新fname（第11~15行）；最终将代码文件持久化至集合F中，并作为F_{out}返回（第16~18行）。

该方法结合YAML配置、惰性模板加载与上下文感知变量绑定，实现从领域特定语言（Domain-Specific Language，DSL）到目标代码的自动转换，具备灵活、高效和安全的优势，并通过MISRA-C预验证确保生成代码的规范性与可靠性。

算法2. 基于元数据驱动的多态代码生成算法

输入：配置规约C_{cfg}，模板集合T_{tpl}，元数据模型M_{meta}
输出：生成文件集合F_{out}

初始化持久化上下文 $F \leftarrow \varnothing$
解析配置 $C_{ir} \leftarrow$SemanticParse (C_{cfg})//生成中间表示
加载模板 $T_{ast} \leftarrow$LoadWithLazyInit (T_{tpl})
for each $tpl_i \in T_{ast}$ do
  if ¬ TypeCheck (tpl_i,C_{ir}.ω_k) then
    LogDiagnostic (DIAG_TEMPLATE_INCOMPATIBLE, tpl_i)
    continue
  end if
  构建上下文 CTX $\leftarrow$Merge (M_{meta}, C_{ir}.ϕ_n)
  执行部分求值 $tpl_{pe} \leftarrow$PartialEval (tpl_i,CTX)
  生成中间代码 code $\leftarrow$TemplateExpand (tpl_{pe})
  计算文件名 fname $\leftarrow$ApplyNamingPolicy (C_{ir}.ϕ_n,tpl_i)
  if FileExist (fname) $\wedge$ ¬C_{ir}.τ_m.overwrite then
    fname $\leftarrow$ApplyVersioning (fname, SHA1(code))
  end if
  $F \leftarrow F \cup$ { Persist (fname, code)}
end for
return F

图11　基于元数据驱动的多态代码生成算法

4　测试验证

为验证XBus消息总线生成器在代码质量、运行性能和工程适用性等方面的综合表现，本文从软件单元验证、系统集成测试和开发效率3个维度构建了完整的测试体系。

4.1　软件单元验证

4.1.1　静态质量分析

本文采用结构质量、类型安全、代码规范与安全漏洞四维质量评估体系对生成器代码进行了全面验证，测试结果如表2所示。除代码规范达标率为98.7%，其余指标均为100%，显示生成器具备良好的结构设计质量和安全性，达到生产级要求。

表2　静态质量评估结果

维度	评估工具	合规标准	达标率/%
结构质量	Radon	圈复杂度≤15	100
类型安全	Pyright	类型错误=0	100
代码规范	Pylint	PEP8违规≤5%	98.7
安全漏洞	Bandit	高危漏洞=0	100

4.1.2　动态测试

为验证生成器关键模块的运行时特性，本文从正确性、时延和容错性3个方面开展测试，结果如表3所示。信息提取器和代码生成器在常规和异常输入下均实现100%通过率；完整流程在复杂输入和故障条件下保持100%稳定性，在80%资源故障率下，系统核心功能未受影响，满足工业级实时嵌入式软件要求。

4.2　软件集成测试

4.2.1　测试环境

如图12所示，为验证生成器在目标平台的集成性能，搭建了基于NXP S32K3网关ECU的硬件在环测试平台。系统软件栈采用AUTOSAR CP架构，ASW由Simulink生成，BSW集成iSOLAR与EB工具链，RTE由本工具生成的FreeRTOS适配代码实现。使用S32DS与J-Link完成烧录调试，通过USB-CAN进行总线验证。

表3　关键模块动态测试结果

测试对象	输入类型	指标	通过率/%
提取器	常规/异常	正确性/容错性	100
生成器	常规/异常	正确性/鲁棒性	100
完整流程	复杂/故障	时延/容错	100

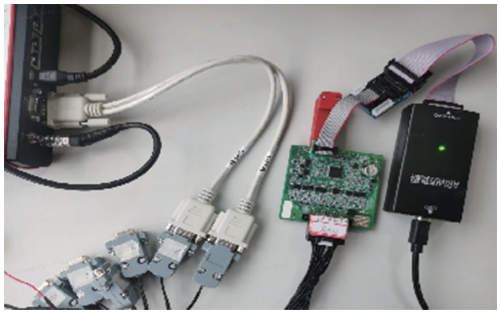


图12　联调测试环境

4.2.2　系统初始化测试

在S32DS中集成生成的RTE与整车软件后编译，编译零错误/零警告，烧录后系统稳定运行。自动生成代码行数达28 976行，运行通过率100%，验证其工程集成与编译兼容性良好，符合车规级质量要求。

4.2.3　功能集成测试

本测试验证生成代码在真实场景下的功能完备性，覆盖通信接口、数据交互等核心功能，具体测试内容包括：

a. RTE接口。如表4所示，覆盖S/R、C/S及模式切换共5类接口，312项测试均100%通过。

b. 外部通信验证。通信数据帧正确率100%，平均延迟(68±22) ms，满足220 ms性能要求（见表4）。

c. IO数据交互。如表5所示，数字信号识别准确率100%，模拟信号误差≤0.5%，均符合设计指标。

表4　RTE接口功能测试概览

接口类别	测试接口	测试要点	结果/%
S/R隐式	Rte_IRead等	缓存读写准确性	100
S/R显式	Rte_Read等	指针操作与一致性	100
C/S同步	Rte_Call	参数处理与同步性	100
C/S异步	Rte_Result等	非阻塞与响应性	100
模式切换	Rte_Switch等	状态切换与实时性	100

结果表明，生成器代码在通信可靠性、实时响应与数据处理方面性能稳定，具备量产部署能力。

表5　通信与IO测试性能指标

测试类	关键指标	实测值	达标要求
CAN通信	数据帧正确率/%	100	100
	平均传输延迟/ms	68	<220
IO交互	数字信号识别率/%	100	100
	模拟信号误差率/%	≤0.5	≤1

4.3　开发效率基准测试

为量化化码生成工具对开发流程的优化效果，本研究设计了对照试验：试验组采用代码生成器生成FreeRTOS适配代码（15个功能模块），对照组由3名工程师（平均经验3年）人工实现相同功能。关键效率指标对比如表6所示，测试结果表明：

a. 高效性。代码生成阶段的效率提升显著，单模块开发周期由平均8.2人时缩短至0.5人时，提速15.4倍。

b. 一致性。工具生成的接口代码实现零一致性错误，相较人工开发减少3.4次平均调试迭代。

c. 复用性。跨平台适配效率较人工方法提升12.7倍。

表6　开发效率对比分析　　人时/模块

指标	实验组	对照组	效率提升
代码编写时间	0.5	8.2	15.4倍
调试迭代次数	1.2	6.7	5.6倍
接口一致性错误	0	3.4	完全消除
跨平台移植时间	0.3	4.1	12.7倍

4.4　测试结论

本文围绕XBus消息总线生成器的代码质量、系统集成性能及开发效率，构建了完整的测试验证体系。测试结果表明：

a. 代码质量优良。工具通过静态分析与动态测试，结构规范、类型安全，未发现漏洞，质量达业界最佳实践标准。

b. 系统集成稳定。RTE代码在AUTOSAR CP架构下成功运行，功能接口100%通过验证，通信与I/O交互性能满足实时性与准确性要求。

c. 开发效率显著提升。相比传统开发，单模块开发效率提升15.4倍，跨平台适配效率提升12.7倍，显著降低人工错误与调试成本。

综上，XBus生成器具备高质量、高性能与高效率，能够满足智能汽车嵌入式软件的功能安全与工程应用需求，为FreeRTOS生态下的RTE代码自动化生成提供了高效、可靠的解决方案。

5　结束语

本文开发了一种适配FreeRTOS的RTE代码自动生成工具，采用模块化设计集成了信息提取器、XBus解析器、处理逻辑模板、OS配置工具和代码生成器等模块，构建了高效的XBus消息总线生成器，实现了从ECU描述文件到符合AUTOSAR规范的RTE代码的自动生成。测试结果表明，该工具具备高质量与可靠性，有效解决了传统工具在适配FreeRTOS方面的不足，并显著提升了开发效率和代码资产的复用性。工具创新性地引入了XBus消息总线架构，简化了系统通信接口定义与数据处理流程，同时通过FreeRTOS的开源特性降低了开发成本。未来研究将进一步优化工具性能，探索与AUTOSAR AP架构的结合，适应汽车智能化进程中的多核处理器及新型互连技术，并扩展其在自动驾驶领域的应用，为智能汽车产业提供更坚实的技术支撑。

参　考　文　献

[1] ZHENG B, YAO C. Automobile Profession Responds to the Development of Automobile Intelligence[C]// 2023 8th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA). Chengdu, China: IEEE, 2023: 419-423.

[2] 邹渊, 孙文景, 张旭东, 等. 智能网联汽车多域电子电气架构技术发展研究[J]. 汽车工程, 2023, 45(6): 895-909.
ZOU Y, SUN W J, ZHANG X D, et al. Study on the Technology Development of Multi-Domain Electrical and Electronic Architecture for Intelligent Networked Vehicles[J]. Automotive Engineering, 2023, 45(6): 895-909.

[3] STARON M. Automotive Software Architectures[M]. Cham: Springer, 2021.

[4] KUGLER L. Making Traffic a Thing of the Past[J]. Communications of the ACM, 2022, 66(1): 19-20.

[5] POPOVIĆ V T, VULIĆ M, DAVIDOVIĆ A, et al. Modeling and Development of AUTOSAR Software Components[C]// 2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT). Ancona, Italy: IEEE, 2019: 313-316.

[6] RAN Z, YAN H, ZHANG H M, et al. Approximate Optimal AUTOSAR Software Components Deploying Approach for Automotive E/E System[J]. International Journal of Automotive Technology, 2017, 18(6): 1109-1119.

[7] AUTOSAR. Specification of RTE Software[EB/OL]. (2024-11-27) [2025-05-09]. https://www. autosar. org/fileadmin/standards/R24-11/CP/AUTOSAR_CP_SWS_RTE.pdf

[8] 汪志鸿, 于德营, 马天泽, 等. 车用操作系统技术现状及发展趋势[J]. 汽车工程, 2023, 45(6): 910-921.
WANG Z H, YU D Y, MA T Z, et al. Technical Status and

Development Trend of Automotive Operating System[J]. Automotive Engineering, 2023, 45(6): 910-921.

[9] 张可可，周沛泽．汽车控制器软件开发模式调研[J]. 汽车实用技术，2021, 46(1): 63-66.
ZHANG K K, ZHOU P Z. A Survey on the Development Model of Automotive Controller Software[J]. Automotive Practical Technology, 2021, 46(1): 63-66.

[10] KHENFRI F, CHAABAN K, CHETTO M. Efficient Mapping of Runnables to Tasks for Embedded AUTOSAR Applications[J]. Journal of Systems Architecture, 2020, 110.

[11] BELLA G, BIONDI P, COSTANTINO G, et al. Designing and Implementing an AUTOSAR-Based Basic Software Module for Enhanced Security[J]. Computer Networks, 2022, 218.

[12] SMITH S, KHALID M A S. Automated Generation and Integration of AUTOSAR RTE Configurations[C]// 2022 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). Halifax, NS, Canada: IEEE, 2022: 141-144.

[13] ZHANG M M, TENG Y, KONG H, et al. Automatic Modelling and Verification of AUTOSAR Architectures[J]. Journal of Systems and Software, 2023, 201.

[14] WEI J Y, GAO J J, MA W F, et al. An AUTOSAR-Based Framework for Multi-Core Communication System [C]//2024 2nd International Conference on Intelligent Metaverse Technologies & Applications (iMETA). Dubai, United Arab Emirates: IEEE, 2024: 214-221.

[15] SU W X, XU S J, LIU F. Design of Collaborative Computing Mining Truck Automatic Driving System Based on AUTOSAR Adaptive Platform[C]// 2022 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI). Shijiazhuang, China: IEEE, 2022: 618-622.

[16] 郑燚，张世民，陆云龙．基于AUTOSAR的域控制器以太网通信系统设计实现[J]. 汽车工程，2023, 45(6): 965-973.
ZHENG Y, ZHANG S M, LU Y L. Design and Implementation of Ethernet Communication System on ADAS Domain Controller Based on AUTOSAR[J]. Automotive Engineering, 2023, 45(6): 965-973..

[17] 邱美涵，王晓琳，卞皓．基于AUTOSAR的电动汽车驱动电机控制系统设计与实现[J]. 汽车工程，2018, 40(6): 659-665.
QIU M H, WANG X L, BIAN H. Design and Implementation of the Control System for the Traction Motor of Electric Vehicles Based on AUTOSAR[J]. Automotive Engineering, 2018, 40(6): 659-665.

[18] MITTAPALLI J S, ARTHUR M P. Survey on Template Engines in Java[J]. ITM Web of Conferences, 2021, 37.

（责任编辑　天下溪）

修改稿收到日期为2025年7月7日。

《汽车工程师》征稿启事

《汽车工程师》于1974年创刊，是由中国第一汽车集团有限公司主管、中国第一汽车股份有限公司主办的国内外公开发行的汽车应用技术类学术期刊，为RCCSE中国准核心学术期刊、CACJ中国应用型扩展期刊，被哥白尼精选数据库、EBSCO学术数据库、欧洲学术出版中心（EuroPub）数据库、J-Gate数据库收录。

《汽车工程师》以报道汽车产品设计领域的技术研究成果为主，专注于技术开发与应用层面的创新成果，提供汽车设计创新解决方案。《汽车工程师》将把握电动化、智能化、网联化、共享化的汽车技术主流发展趋势，努力吸收优质稿源，为我国汽车工程技术创新能力提升贡献力量。重点征稿方向包括：锂离子电池与新体系电池、动力电池热管理与热失控、燃料电池及其关键零部件、车用电机及其控制器、自动驾驶汽车环境感知与感知融合、自动驾驶汽车定位与建图、自动驾驶汽车测试、人工驾驶-无人驾驶车辆混行条件下的自动驾驶控制、智能线控底盘、一体化底盘、功能安全和预期功能安全。

热忱欢迎汽车行业的专家学者不吝赐稿，反映国家重点扶持项目、自然科学基金项目和其他重点项目等研究成果的稿件将优先发表，我们期待与您共同践行“把论文写在祖国大地上”的指示精神，为强大中国汽车工业作贡献！

本刊使用协同采编系统处理稿件，请登陆期刊网站注册后投稿，投稿要求见本刊网站中“下载中心”栏目的作者指南，网址：http://tjqc.cbpt.cnki.net。

《汽车工程师》编辑部