ISSN: 3065-1220 Volume 2, Number 3, June 2025

Hans汉斯

嵌入式技术与智能系统

Embedded Technology and Intelligent Systems

Qian Ru Shi Ji Shu Yu Zhi Neng Xi Tong 2025年6月2卷3期

https://www.hanspub.org/journal/etis

Editorial Board

编委名单

ISSN: 3065-1220

https://www.hanspub.org/journal/etis

Embedded Technology and Intelligent Systems, Vol.2, No.3, 133-195
嵌入式技术与智能系统, 2 卷 3 期
Published Online June 2025 in Hans (https://www.hanspub.org/journal/etis)

TABLE OF CONTENTS
目　　录

期刊信息
期刊中文名称：《嵌入式技术与智能系统》
期刊英文名称：**Embedded Technology and Intelligent Systems**
期刊缩写：**ETIS**
出刊周期：双月刊
语　　种：中文

出版机构：汉斯出版社(Hans Publishers, https://www.hanspub.org/)
编辑单位：《嵌入式技术与智能系统》编辑部
主　　编：何立民，北京航空航天大学教授
网　　址：https://www.hanspub.org/journal/etis

订阅信息
订阅邮箱：sub@hanspub.org
订阅价格：180 美元每年

广告服务
联系邮箱：adv@hanspub.org

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(3), 133-148
Published Online June 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.23011

Hans汉斯

基于数据驱动的图像分辨率提升理论和技术综述

刘卫玲[1,2]，侯晓奎[1,2]，常晓明[2*]

[1]太原理工大学人工智能学院，山西 晋中
[2]太原理工大学晓明研究室，山西 太原

收稿日期：2025年7月4日；录用日期：2025年10月15日；发布日期：2025年10月27日

摘　要

在大数据与深度学习浪潮的推动下，数据驱动模型已全面取代传统规则式算法，成为提升图像分辨率的核心引擎。本文综述了数据驱动模型在图像分辨率提升方面的应用，涵盖了三维重建、压缩感知、单像素成像和超分辨率技术，并进一步探讨数据驱动模型在视觉成像、工业无损评估和医学影像处理等实际场景中的落地实践，以及未来的发展趋势。

关键词

数据驱动模型，图像分辨率提升，深度学习

A Survey of Theories and Techniques for Data-Driven Image Resolution Enhancement

Weiling Liu[1,2], Xiaokui Hou[1,2], Xiaoming Chang[2*]

[1]College of Artificial Intelligence, Taiyuan University of Technology, Jinzhong Shanxi
[2]Xiao Ming Research Laboratory, Taiyuan University of Technology, Taiyuan Shanxi

Received: July 4, 2025; accepted: October 15, 2025; published: October 27, 2025

Abstract

Driven by the wave of big data and deep learning, data-driven models have completely replaced

*通讯作者。

文章引用: 刘卫玲, 侯晓奎, 常晓明. 基于数据驱动的图像分辨率提升理论和技术综述[J]. 嵌入式技术与智能系统, 2025, 2(3): 133-148. DOI: 10.12677/etis.2025.23011

traditional rule-based algorithms and become the core engine for improving image resolution. This paper reviews the application of data-driven models in image resolution improvement, covering 3D reconstruction, compressed sensing, single-pixel imaging, and super-resolution technology. The implementation practice of data-driven models in actual scenes such as visual imaging, industrial non-destructive evaluation, and medical image processing, and the future development trend are further discussed.

Keywords

Data-Driven Model, Image Resolution Enhancement, Deep Learning

1. 引言

随着大数据和人工智能时代的到来，视觉数据处理技术经历了从基于规则到数据驱动的范式转变，为工业自动化、医学诊断等多个领域带来了革新。传统的分析方法依赖于手工特征和统计模型，在泛化能力、适应复杂数据分布以及真实世界成像条件下的鲁棒性方面存在局限性。而机器学习，尤其是深度学习的兴起，使得模型能够从海量数据集中自主学习层次化特征，实现了在复杂视觉分析任务中前所未有的准确性和鲁棒性。

数据驱动模型的广泛应用与计算硬件的进步、算法创新以及大规模标注数据集的可用性密切相关。早期机器学习模型如支持向量机和决策树展示了利用数据进行模式识别的潜力，但其浅层架构难以捕捉高维图像数据中的复杂空间和语义关系。卷积神经网络的兴起，通过卷积层、池化操作和非线性激活引入了层次化特征提取，开创性的工作如 AlexNet 和 ResNet 展示了 CNN 在图像分类中超越人类水平性能的能力，为其在目标检测、分割等领域的主导地位铺平了道路。

近年来，Transformer 架构的出现进一步颠覆了计算机视觉领域，其自注意力机制在建模图像中的长距离依赖和全局上下文方面表现出色，解决了 CNN 的局部性限制。视觉 Transformer (ViT)和检测 Transformer (DETR)在需要整体理解的图像描述和全景分割等任务中取得了最先进的结果。与此同时，生成模型如去噪扩散概率模型(DDPM)的出现，通过模拟扩散过程为图像重建提供了新的框架。

尽管取得了这些进展，但仍存在重大挑战。首先，退化建模不足，真实场景中的噪声、模糊类型远比合成训练数据复杂，导致模型在实际应用中表现失准；其次，计算成本高昂，高参数量的 Transformer 等架构对显存与推理时间要求极高，难以在边缘设备或实时系统中部署；第三，多模态融合薄弱，RGB、深度、红外等互补信息尚未被有效协同利用，限制了分辨率提升的上限；最后，评价指标失准，传统 PSNR/SSIM 常与主观视觉质量脱节，难以真实反映人眼对细节、纹理及整体清晰度的感知偏好。

本文聚焦图像分辨率提升的数据驱动模型，涵盖从基础技术到最新创新，系统综述三维重建、压缩感知、单像素成像及超分辨率技术。通过研究它们在视觉检测、工业无损检测(NDT)和医学成像中的应用，我们突出了这些模型的变革潜力和未解决的挑战。

本工作的贡献有三个方面：

1. 技术综合：统一传统方法(压缩感知/字典学习)与深度学习(CNN/Transformer/DDPM)的分辨率提升框架。

2. 退化–重建关联分析：揭示采样策略(如单像素成像)、退化模型与重建质量的耦合机制。

3. 应用场景适配：结合工业微缺陷检测、医学低剂量成像等需求提出优化路径，为研究人员和从业者提供了可操作的见解。

本文的其余部分安排如下：第 2 节专注于图像分辨率增强，包括三维重建、压缩感知和超分辨率。第 4 节讨论跨行业的应用，在第 5 节和第 6 节分别进行比较分析和未来趋势的探讨。通过这种结构，我们旨在为读者提供数据驱动视觉处理在图像分辨率提升方面的全面理解，促进理论和应用领域的创新。

2. 图像分辨率增强

2.1. 三维重建技术

三维重建技术通过结合图像采集、特征提取、匹配和模型优化等步骤，从二维图像中恢复三维场景结构，增强图像的立体感和细节。随着技术进步，该技术已发展到利用深度学习进行高效准确的重建，并在文化遗产保护、娱乐、医疗、机器人导航、工业设计和虚拟现实等多个领域得到应用。它分为基于传统多视图几何和基于深度学习的算法，正朝着深度学习、多模态融合、实时重建和交互式模型等方向进步。基于图像的三维重建技术可以分为基于传统多视图几何的三维重建算法和基于深度学习的三维重建算法。

2.1.1. 基于传统多视图几何的三维重建算法

立体视觉技术通过分析多视角图像获取深度信息以重建三维模型，分为依赖自然特征的被动方法和通过投射已知模式测量距离的主动方法。传统三维重建算法如结构光重建(SFM)和多视角立体(MVS)，通过图像配准、视差计算和特征匹配恢复三维结构，但受光照条件影响且精度有限。

从 2013 年到 2021 年，研究者们提出了多种 SFM 算法，如全局 SFM [1]、在线服务平台[2]、COLMAP 增量 SFM [3]、HSFM [4]以及基于增量 SFM 的单目三维重建方法[5]，这些技术提高了大规模三维重建的鲁棒性、精度和可扩展性。

SFM 通过特征点匹配获取相机参数，但产生稀疏点云。提供更详尽的三维信息，其中稠密点云重建 MVS 基本流程图见图 1。MVS 则通过像素级匹配生成更密集的点云，提供更详尽的三维信息。MVS 自 2006 年以来也经历了重要发展，Seitz 等人的对 MVS 算法的系统性介绍[6]、Sinha 等人的基于体素的 MVS 方法[7]，以及 Lin 等人结合双目立体视觉和特征匹配的三维重建方法[8]，推动了 MVS 领域的发展。

传统多视图几何三维重建依赖相机采集，精度不及激光点云。相机类型影响结果：彩色相机提供颜色信息但受光照影响；红外相机不受光照影响，但无法捕获颜色，实际应用时需权衡这些因素。

2.1.2. 基于深度学习的三维重建算法

传统的三维重建技术，如 SFM 和 MVS，依赖亮度一致性，在理想环境下效果良好，但在纹理弱或高反射环境下易出现不准确或空洞。深度学习方法通过编码解码过程，无需复杂校准，有效改善了这些问题。

2021 年，研究者们提出了基于深度神经网络的三维重建技术[9]-[11]，通过无监督学习、深度特征测量和增量 SFM 结构的深度融合，提高了三维重建的准确性和鲁棒性。

自 2014 年 Eigen 等人[12]首次将 CNN 应用于三维重建以来，深度学习在三维重建领域取得了显著进展，如 2015 年的多任务 CNN [13]、2017 年的 Pix2Face [14]和后续的 MVSNet 系列[15]-[20]，这些技术通过引入先进的神经网络架构，显著提升了三维重建的精度和效率。

NeRF 技术的出现推动了三维场景隐式表示的发展，从 2020 年的全连接神经网络到 2023 年的高分辨率数据训练[21]-[24]，NeRF 不断优化，提升了重建质量和细节。同时，Stucker 和 Schindler [25]、Peng 等人[26]的技术展示了深度学习在动态场景合成和多视角重建中的应用，而 Huang 等人[27]的方法则通过

二维卷积网络与三维神经网络辐射场的交互学习，为三维场景重建带来了高质量的风格化效果。这些研究证明了深度学习在解决传统三维重建挑战，如表面空洞问题方面的潜力。

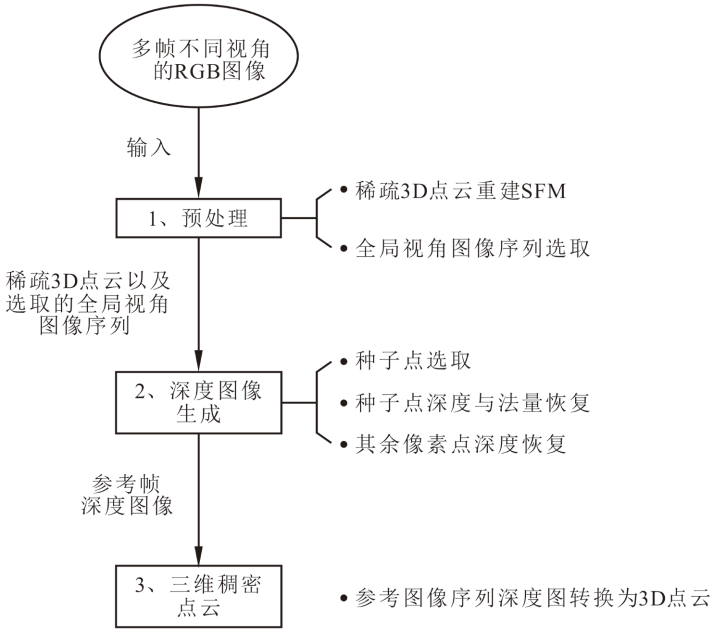

Figure 1. Basic flow chart of MVS reconstruction from dense point clouds
图 1. 稠密点云重建 MVS 基本流程图

2.1.3. 基于学习的点云稠密化方法

随着深度学习的发展，研究者提出了一系列端到端的网络，可直接由稀疏点云输出高密度表示，无需显式三维重建。2018 年的 PU-Net [28]率先用特征提取与多层感知器逐层生成新点；其后续 PU-GAN [29]、Dis-PU [30]等进一步引入生成对抗网络与几何约束，提升点的均匀性与几何一致性。另一类工作利用 NeRF 变体[31]等隐式神经表示，把点云编码为连续隐式场，让网络预测任意位置的几何概率，实现任意分辨率稠密化。为了补全缺失细节，多模态方法融合 RGB、深度或法向信息，通过跨模态注意力机制联合图像纹理与点云几何，实现高保真增强[32]。针对标注数据稀缺的现实，最新研究还探索了基于重建一致性、几何不变性的自监督或无监督生成模型，显著提高了算法在真实场景中的适应性[33]。

2.2. 压缩感知

压缩感知(Compressive Sensing，简称 CS)是一种信号处理理论，其基本流程图见图 2，它提出了一种革命性的采样方法，允许从远低于奈奎斯特率的测量值中恢复稀疏或可压缩信号。这一理论的核心在于，如果一个信号在某种域(如时间域、空间域或频率域)中是稀疏的，那么它可以通过远少于传统采样定理要求的样本数来重建。

在图像感知与压缩领域，压缩感知的概念尤其具有吸引力。图像和视频信号通常具有内在的冗余性，这意味着它们在变换域(如小波变换或傅立叶变换)中只有少数几个系数是显著的，而其他许多系数接近于零。利用这一特性，压缩感知技术可以在保持图像质量的同时，大幅度减少需要存储或传输的数据量。例如，Yoshida 等[34]探索了将人类视觉感知引入图像压缩感知问题的可能性，通过将视觉显著性与几何特征相结合，构建视觉启发的“重要性图”，指导压缩采样过程，并结合深度图像先验(DIP)与嵌入空间流形建模(MMES)在解码端重建图像。实验结果表明，该方法在极低采样率下仍能保留关键的视觉特征，

显著优于传统的随机或均匀采样策略，为压缩感知领域带来了新的感知驱动思路。

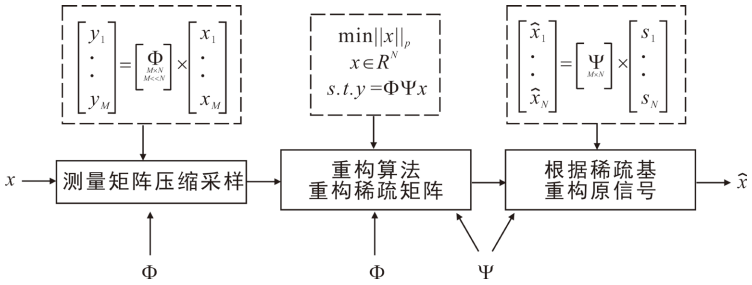

Figure 2. Basic flow chart of compressed sensing
图 2. 压缩感知基本流程图

2.2.1. 压缩感知框架

压缩感知框架是压缩感知技术的核心，它定义了如何从少量测量值中重建原始信号。压缩感知框架的核心思想是利用信号的稀疏性或可压缩性来减少数据采集量，从而实现高效的数据采集和重建。它突破了奈奎斯特采样定理的限制，允许在远低于信号带宽的速率下进行采样，并通过算法重建出高质量的信号。He 等[35]提出扩散自适应框架，通过将测量矩阵分布式存储于网络节点并引入扩散 l_0-LMS 与 mini-batch 扩散算法，实现了稀疏信号的协同快速重建，在收敛速度与重建精度上均优于单机 l_0-LMS。Oikonomou 等[36]提出的一种基于变分贝叶斯框架的新型压缩感知算法，实验表明该方法在多种场景下均优于现有主流算法。这些框架的成功应用，证明了压缩感知在图像重建和压缩方面的巨大潜力。

2.2.2. 字典学习

字典学习是压缩感知中一个重要的技术，它通过学习信号的稀疏表示，从而实现高效的压缩和重建。字典学习的目标是从训练数据中学习出一组基函数，使得每个信号都可以表示为这些基函数的线性组合，并且组合系数尽可能稀疏。稀疏性意味着只有少数几个系数是非零的，这有助于降低数据的存储和传输成本。S Li 等[37]提出基于稀疏编码的双字典超分辨率框架，通过 Gabor 滤波提取多尺度特征并引入残差字典补偿高频细节，显著提升了微铣刀具磨损图像的分辨率与监测精度。

2.2.3. 算法

压缩感知算法是信号重建的关键，它决定了重建的精度和效率。例如，Beck 等[38]提出了一种快速迭代收缩阈值算法(FISTA)，并将其应用于图像去模糊问题，取得了显著的性能提升，验证了 FISTA 在图像检测与压缩感知领域应用的有效性。Li 等[39]提出了一种基于总变分正则化的高效算法 TVAL3，并将其应用于单像素相机图像重建，显著提高了图像检测与压缩感知的性能，验证了 TVAL3 在图像恢复方面的有效性。表 1 为 FISTA 与 TVAL3 的相关比较。

Table 1. Relevant comparison between FISTA and TVAL3
表 1. FISTA 与 TVAL3 的相关比较

	收敛速度	内存占用	重建质量	适用数据类型
FISTA	快	低	高	稀疏信号(MRI)
TVAL3	中等	中等	极高	单像素成像

2.3. 不同图像的稀疏性

在图像处理和计算机视觉领域，稀疏性(Sparsity)是一个重要的概念，它指的是在表示图像时，图像

的某些属性或特征在某种域中只有少数几个非零元素，这些非零元素可以捕捉到图像的主要信息。图像的稀疏性是自然存在的，许多类型的图像实际上在诸如小波变换、傅立叶变换或 DCT (离散余弦变换)等正交变换下呈现高度稀疏。通过寻找最合适的稀疏表示，我们能够捕捉到图像的关键特征，同时减少冗余信息，这对于图像编码、压缩、分类和重建等方面都有积极影响，图像的稀疏表示方法分为以下几类。

2.3.1. 贪婪策略近似

贪婪策略近似方法通过迭代选择字典中与图像残差最匹配的原子，并利用最小二乘法计算原子系数，以近似求解稀疏表示问题。匹配 pursuit (MP)算法和正交匹配 pursuit (OMP)算法是最典型的贪婪策略近似方法，它们通过选择最佳匹配原子来逐步逼近图像的稀疏表示例如，Shengqin Bian 和 Lixin Zhang [40]比较了匹配追踪算法在图像重建中的应用性能，结果表明，在无噪声信号重建中，子空间追踪算法表现略优于其他算法；在二维图像重建中，当压缩比低时，子空间追踪算法也表现更佳。此外，还有许多改进的贪婪算法，如 ROMP、CoSaMP、StOMP、SP、SAMP、TMP、TBOMP、FBP 等，它们在效率、鲁棒性和收敛速度等方面进行了优化。Rajaei Khatib 等[41]提出了一种名为学习贪婪方法(LGM)的新型神经网络架构，用于稀疏编码和图像检测，该架构能够有效地学习图像的特征表示，并取得了优于传统方法的性能。

2.3.2. 约束优化策略

约束优化策略将稀疏表示问题转化为约束优化问题，并利用高效的优化方法求解。梯度投影稀疏重建(GPSR)算法将原始的无约束非光滑问题转化为一个光滑的可微约束优化问题，利用梯度下降和标准线搜索方法求解 L1 正则化问题，Li 等[42]将其与 Krylov 子空间投影相结合，应用于三维电阻层析成像(ERT)，在保持高精度图像重建的同时显著降低了计算耗时，验证了 KGPSR-BB 在实时稀疏正则化与工业过程成像中的有效性。基于内点法的稀疏表示策略，如 TNIPM 算法将 L1 范数最小化问题转化为一个无约束光滑问题，利用截断牛顿法和内点法求解 L1 正则化问题，适用于大规模稀疏表示问题。交替方向法(ADM)算法可以有效地求解 L1 正则化问题的对偶问题，进一步提高算法的效率。Yan Yang 等[43]提出了一种基于交替方向乘子法(ADMM)的深度学习架构，并将其应用于图像压缩感知，取得了优于传统方法和深度学习方法的效果，验证了深度学习在图像检测与稀疏表示方面的有效性。

2.3.3. 基于近端算法的优化策略

基于近端算法的优化策略利用近端算子迭代求解稀疏表示问题，如 ISTA、FISTA、SpaRSA、ALM 等。这些算法将 L1 正则化问题转化为 L2 最小化问题，并利用收缩算子或投影算子进行迭代求解。例如，ISTA 算法利用收缩算子迭代求解 L1 正则化问题，具有较高的收敛速度和计算效率。FISTA 算法是 ISTA 算法的改进，利用 Lipschitz 常数近似 Hessian 矩阵，并加速收敛，Li 等[44]将其与单频全息成像结合，提出 SFH-FISTA 实现 3D 稀疏毫米波成像，实验显示在 50%采样率下仍高质重建，验证了 FISTA 在雷达稀疏成像中的高效与稳健。SpaRSA 算法利用自适应连续和 BB 奇异值方法优化 L1 正则化问题，在图像压缩领域展现出高效性、灵活性、稳定性和快速收敛的优势，使其在处理大规模稀疏信号恢复问题时表现出色，尤其适用于不同类型的图像数据。ISTA、FISTA、SpaRSA 与 ALM 的相关比较见表 2。

2.3.4. 基于同伦算法的稀疏表示

基于同伦算法的稀疏表示方法利用同伦算法追踪 L1 正则化问题中参数 λ 的变化路径，逐步更新稀疏解。例如，LASSO (最小绝对收缩和选择算子)同伦算法和 BPDN (基追踪去噪)同伦算法分别追踪 LASSO 和 BPDN 问题中参数 λ 的变化路径，逐步更新稀疏解，LASSO 更多地被统计学界使用，而 BPDN 则更多地被信号处理界采用。在实际应用中，当观测数据可能包含噪声时，这两种方法都需要进行适当地调整

以处理噪声问题。此外，还有基于同伦的迭代重新加权 L1 最小化算法，它利用同伦算法更新 L1 正则化问题的权重，进一步提高算法的效率，Sining Huang 等[45]提出了一种基于扩展重加权 ℓ1 最小化算法(ERMA)的图像恢复方法，有效提高了图像恢复的信号噪声比(SNR)、结构相似性(SSIM)和均方误差(MSE)，并通过仿真实验验证了其在图像检测和稀疏表示方面的优越性。

Table 2. Related comparison of ISTA, FISTA, SpaRSA, and ALM
表 2. ISTA、FISTA、SpaRSA、ALM 的相关比较

	收敛速度	计算复杂度	扩展性
ISTA	O (1/k)	低	仅适用于凸问题
FISTA	$O(1/k^2)$	低	仅适用于凸问题
SpaRSA	自适应步长加速	中低	支持非光滑问题，部分非凸扩展
ALM	超线性或线性收敛	较高	可处理非凸问题

2.4. 单像素成像

单像素成像(Single-Pixel Imaging, SPI)是一种颠覆传统的成像技术，它不依赖于成像设备上每个像素点的独立探测，而是利用单个像素探测器来捕获整个场景的图像信息。这种技术的核心在于，通过空间光调制器(Spatial Light Modulator, SLM)或者类似的设备对照明光束进行调制，产生一系列具有特定模式的照明图案，这些图案依次投射到目标物体上[46]。SPI 技术的发展，特别是在运动目标成像、盲重建、图像加密和隐藏以及照明图案优化等领域，为我们提供了新的视角和方法。

2.4.1. 运动物体成像

在运动物体成像领域，单像素成像(SPI)技术因其独特的成像机制而展现出巨大潜力，同时也面临着一系列挑战。其中最主要的挑战之一是如何在保持图像质量的同时提高成像速度。这是因为 SPI 技术在获取图像时通常需要进行多次的照明图案投射和相应的信号采集，这个过程在面对快速移动的物体时会变得尤为困难。Monin 等[47]提出一种基于循环采样矩阵与多帧运动估计的单像素成像算法，通过直接在投影域检测并补偿全局或局部运动，在目标运动过程中仍能实现高保真重建，为动态场景下的单像素成像提供了实时、稳健的解决方案。

2.4.2. 盲重建

盲重建(Blind Reconstruction)是一个在信号处理和图像分析领域常见的概念，它指的是在不知道原始信号或图像具体参数或特性的情况下，仅通过观测到的数据来恢复原始信号或图像的过程。在盲重建领域，Zhuang 等[48]提出了一种结合深度图像先验(DIP)和结构化深度神经网络的方法，用于解决盲图像去模糊(BID)问题，并在未知核大小和显著噪声的情况下表现出稳定性，验证了该方法在提高图像去模糊效果方面的有效性。Song 等[49]提出了一种基于粉噪声散斑与深度学习的计算鬼成像框架，在无需实验训练数据的情况下即可从 0.8% Nyquist 采样率中重建出高保真图像，并在未知系统响应与强噪声条件下实现鲁棒盲重建，验证了该方法在极低采样与复杂环境下的有效性。

2.4.3. 图像加密和隐藏

SPI 技术在图像加密和隐藏方面利用其独特的成像机制提供了新的安全策略，它通过随机相位掩模对图像进行编码，生成难以识别的噪声状图案，从而保护图像内容不被未授权访问。此外，SPI 技术结合混沌理论可以生成高度随机的编码模式，进一步增强图像的安全性。Zhang 等[50]提出了一种基于混沌棕

楝相位掩模(CPPM)和菲涅耳变换(FrT)的光学单通道彩色图像加密方案，并将其应用于光学信息安全领域，取得了显著的安全性提升，验证了该方案在提取光学图像隐藏信息方面的有效性，这为 SPI 在图像加密领域提供了新的视角。

2.5. 人工智能超分辨率及其局限性

超分辨率技术(Super-Resolution，简称 SR)是一种旨在提高图像或视频分辨率的技术。传统的图像放大方法往往会导致图像模糊和失真，而超分辨率技术则通过算法重建图像的细节，生成比原始图像更清晰、分辨率更高的图像，同时保留原始内容和结构，Liu 等[51]提出了一种基于 Cycle-GAN 的超编码分辨率重建方法，通过无配对训练策略在远低于奈奎斯特采样条件下实现 2×超分辨成像，实验验证其在 3.125%~25%采样率下显著提升图像细节与边缘锐度，为低数据量、高保真超分辨率成像提供了新途径。超分辨率技术与人工智能结合的关键优势在于其能够从大量的数据中学习到复杂的特征，并利用这些特征来增强图像的细节和质量。这种结合不仅提高了图像的质量，还扩展了超分辨率技术在多个领域的应用范围，为图像处理领域带来了新的可能性。

2.5.1. 单一图像超分辨率

单一图像超分辨率(SISR)技术是一种图像增强方法，它旨在从单个低分辨率图像中恢复出高分辨率图像。SISR 的挑战在于，由于成像系统的局限性或数据获取过程中的约束，低分辨率图像丢失了高频细节信息。为了克服这些限制，SISR 通常采用先进的算法，如基于学习的方法，特别是深度学习技术。Liang 等[52]提出了一种基于 Swin Transformer 的端到端图像复原框架 SwinIR，通过局部－全局混合注意力机制直接学习从低分辨率到高分辨率的映射，在多个超分辨率任务及退化场景下均显著优于现有 CNN 方法，验证了 Transformer (结构示意图见图 3)在单图像超分辨率中的有效性与高效性。Zhang 等[53]提出了一种由随机模糊－降采样－噪声级联并可随机洗牌的实用退化模型，联合 ESRGAN 架构端到端训练出 BSRGAN，在未知复杂退化的真实图像上实现盲超分辨率，显著提升了视觉质量与鲁棒性，验证了该方法在实际场景中的有效性。

2.5.2. 多图像超分辨率

多图像超分辨率(MISR)技术利用多个低分辨率图像重建一个高分辨率图像。这种方法通常涉及到图像配准，以确保多个图像中的场景对齐，然后通过融合技术合并图像信息以提高分辨率。与单图像超分辨率(SISR)技术不同，MISR 通过整合多幅图像中的互补信息，能够更有效地恢复高分辨率图像的细节和结构。SISR 与 MISR 的相关比较见表 3。

Salvetti 等[54]提出了一种基于残差特征注意力的深度神经网络 RAMS，利用 3D 卷积同时融合多幅低分辨率遥感影像的时空信息，实现 3×超分辨率重建，在公开 Proba-V 数据集上显著优于单图与现有多图方法，验证了其在大尺度遥感场景中的有效性与可迁移性。MISR 的优势在于它可以利用多个视角中的冗余信息来增强细节并减少噪声，这种方法特别适合于场景相对静态的情况，Xiu 等[55]提出的一种新的端到端网络结构 CoT-MISR，结合了卷积和 Transformer 的优势，有效利用低分辨率图像的局部和全局信息，在 PROBA-V 数据集上取得了目前多图像超分辨率任务的最佳性能，为遥感图像融合提供了新的思路。随着计算能力的提升和算法的改进，MISR 技术在处理大型图像数据集和提供更高分辨率图像方面变得更加有效。

2.5.3. 局限性

当前人工智能与图像超分辨率的结合取得了一定进展，但仍面临挑战。首先，虽然 CNN 模型被广泛使用，但 GAN 模型在处理复杂场景和不同缩放任务时更具优势，未来的研究需要探索如何结合两者的优

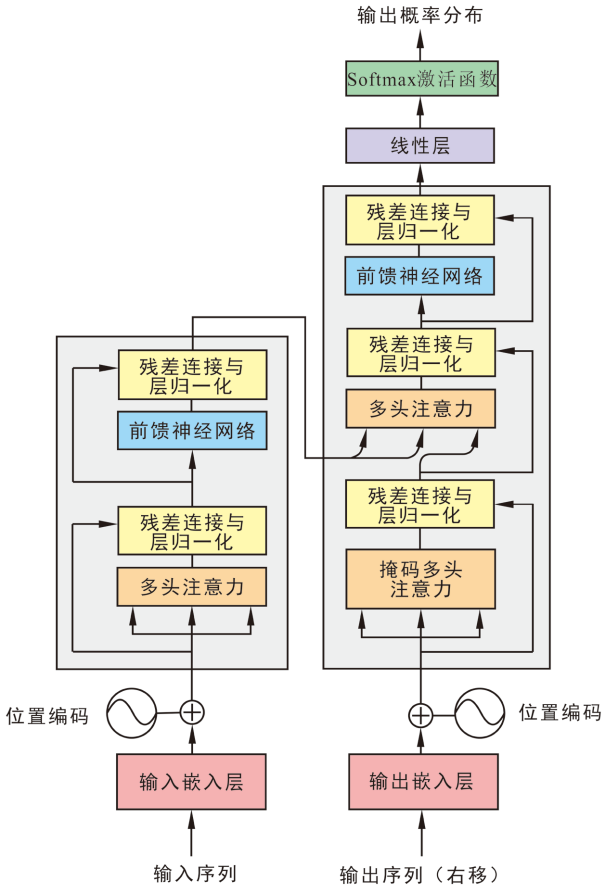

Figure 3. Schematic diagram of the structure of Transformers
图 3. Transformers 结构示意图

Table 3. Correlation comparison between SISR and MISR
表 3. SISR 与 MISR 的相关比较

	输入类型	重建质量	计算成本	典型算法
SISR	单张低分辨率图	中等	低	SRCNN、ESRGAN
MISR	多张低分辨率图	高	高	CoT-MISR、MVSR

势并简化网络结构[56]。其次，现有模型在适应多变的实际场景方面存在困难，需要设计更灵活的模型来利用不同场景的先验知识。此外，可以引入深度学习中的多种学习方法，如注意力机制和多因素学习，以提升模型性能。在图像退化和采样方法方面，需要探索更接近真实情况的模型和方法，以弥补现有方法与实际应用之间的差距。最后，现有的目标函数和评估标准存在一定的局限性，需要开发更合理的损失函数和质量评价方法，以平衡模型的精度和感知质量。

3. 应用案例

3.1. 视觉检测

视觉检测是利用计算机视觉技术来识别、定位和分析图像或视频中的物体和事件。它不仅仅是一个技术过程，更是一种使机器能够理解和解释视觉信息的革命性能力，这种能力使得机器能够在各种复杂

的环境中执行任务。数据驱动模型，尤其是深度学习模型，已经成为实现这一能力的核心工具。这些模型通过从大量图像数据中学习，能够自动提取特征并建立复杂的模式识别系统。深度学习的发展，特别是卷积神经网络(CNN)的广泛应用，已经极大地推动了视觉检测技术的进步。CNN 在图像和视频分析任务中展现出了前所未有的性能，使得机器能够在没有人类直接干预的情况下，准确地识别和定位图像中的物体。

随着技术的不断进步，视觉检测的应用已经扩展到了许多新的领域。在馆藏数字化领域，倪劼等[57]提出一种改进的 Real-ESRGAN 模型，针对馆藏近代低分辨率图像引入多尺度特征融合与通道注意力机制，在 4×超分辨率重建后将图像送入后续视觉检测流程，显著提升了历史文献中文字、图案等关键细节的检出率(PSNR ↑ 3 dB, SSIM ↑ 0.0672)，为图书馆数字化展示与智能检索提供了高保真视觉基础。在体育分析中，视觉检测技术被用来追踪运动员表现和比赛动态，谢竞光和程新年[58]提出的一个结合生成对抗网络(GANs)和循环神经网络(RNNs)的系统，用于预测排球比赛中的战术数据。该系统利用计算机视觉技术自动从比赛视频中提取关键信息，并运用深度学习进行智能分析。这一方法能有效识别和理解排球比赛中的复杂战术行为，并提供准确预测，为战术分析、训练和比赛决策提供技术支持。

尽管视觉检测技术已经取得了显著的成就，但仍有许多挑战需要克服。例如，如何在不同的光照条件和复杂背景中保持高准确性，如何提高模型的实时处理能力，以及如何处理大规模图像数据集。未来的研究将继续探索更高效的算法，提高模型的泛化能力和解释性，以及开发新的硬件和软件解决方案，以满足不断增长的应用需求。

3.2. 工业无损检测

在工业应用中，无损检测(NDT)扮演着至关重要的角色，它确保了产品在不遭受物理损伤的情况下，其质量和安全性得到有效保障。随着与图像检测技术的融合，NDT 的能力和效率得到了显著提升，成为了维护工业产品质量的强有力工具。利用高分辨率成像技术、尖端图像处理算法以及机器学习模型，这些综合技术能够自动化地识别材料表面的微观缺陷，精确评估焊接与连接的质量，持续监测复合材料的结构完整性，并准确验证电子组件的制造精度。

在这一领域，高分辨率成像技术的应用尤为关键，其提供的超高像素密度和细节还原能力，使得检测过程能够捕获材料表面的微观结构。通过提升图像的空间分辨率与对比度，这些技术为后续分析提供了更清晰、更丰富的视觉数据基础，使微米级甚至纳米级的特征可视化成为可能，从而为无损检测的精确性奠定技术前提。

随着图像检测技术的不断发展，其在工业无损检测领域的应用越来越广泛。在窄间隙焊缝的缺陷检测中，Nicolson 等[59]以双串联相控阵超声与 FMC 成像，在窄间隙焊缝中实时分辨亚毫米级未熔合缺陷，为核电厚壁焊接提供高分辨率在线检测，使核电与可再生能源等领域厚壁结构的高质量、低成本制造取得可靠保障。在涡轮叶片 X 射线成像领域，马钟、赵歆波等人[60]提出了一种基于频域亚像素配准与非均匀插值的超分辨率重建技术，该技术利用多幅含相对位移的低分辨率 DR 图像，通过傅里叶域精确配准和双调和样条插值，实现了分辨率提升至原图 4 倍的高保真成像，有效揭示了叶片表面及内部的细微缺陷，实验验证其在航空无损检测中具有显著的工程应用价值。

这些研究表明，高分辨率图像技术的进步在工业无损检测领域正发挥越来越关键的作用。通过持续提升成像设备的解析能力与信噪比，该技术为材料微观结构的可视化提供了更强大的支撑，从而推动无损检测向更高精度和可靠性发展。

3.3. 医疗成像

医疗成像技术是一种用于获取人体内部结构图像的非侵入性或微创性技术，它在临床诊断和治疗中

发挥着至关重要的作用。通过不同的成像原理，如 X 射线、超声波、磁共振、放射性核素等，医疗成像技术通过不断提升空间分辨率和对比度，能够以微米级精度呈现人体内部器官、组织的微观结构，为临床研究提供高保真图像基础。数据驱动模型显著优化了成像设备的解析能力与信噪比，使纳米级生物特征的可视化成为可能，推动医疗成像向更高清晰度和效率发展。

医疗成像技术结合图像检测旨在提高疾病诊断的准确性和效率，高分辨率成像技术(如 CT、MRI)通过提升像素密度和层析精度，为图像处理算法提供了更丰富的结构细节。基于深度学习的超分辨率重建和噪声抑制模型，能进一步优化原始图像的纹理清晰度与边界锐度，使细胞级结构或微血管形态等亚视觉特征得以清晰呈现，为医学研究奠定高质量数据基础。

Sobek 等[61]开发了一个名为 Med-YOLO 的三维医学图像目标检测框架，该框架基于 YOLO 模型。Med-YOLO 通过 3D 版本替换了 2D 神经网络层，使其能够理解和分析医学图像中的三维结构。Zhe Guo 等[62]的多模态分割算法，利用超高分辨率图像叠加，实现了亚毫米级软组织纹理的精准分层映射。此外，AI 医学影像模型如 SLIViT [63]，它作为一种深度学习模型，能够快速高效地进行专家级图像分析，该模型通过融合多尺度分辨率数据，在低信噪比条件下仍能保持生物标记物成像的完整性，验证了高分辨率技术对复杂医学图像分析的普适价值。这些技术的应用，使得医学影像检测更加精确，有助于提高疾病诊断的准确性和治疗的成功率。

医疗成像中的图像检测技术面临的挑战在于平衡辐射剂量与分辨率需求(如低剂量 CT)、突破衍射极限的微观成像，以及海量高分辨率数据的实时处理。未来研究将聚焦开发轻量化超分辨率算法、量子成像传感器等硬件革新，以突破现有分辨率极限，实现无损活体纳米级成像。

4. 比较与讨论

在图像检测领域，传统图像处理方法(如 SIFT、HOG 特征提取)依赖人工设计特征与统计模型(PCA、LDA)，其优势在于算法透明、计算效率高，但在复杂场景(如动态模糊、低纹理区域)中泛化能力有限。而深度学习模型(CNN、Transformer)通过数据驱动的端到端学习，自动挖掘多层次特征表达，显著提升了对噪声、形变等干扰的鲁棒性。

不同领域对图像质量的差异化需求，正推动分辨率增强技术沿着“场景定制”路径快速演进：在工业检测中，时序生成模型通过捕捉视频帧间动态信息，显著改善了运动模糊图像的恢复效果；在医学成像领域，三维分割算法借助超高分辨率数据，突破了压缩感知稀疏重建的精度瓶颈；而在文化遗产保护场景，改进的超分辨率模型针对古籍褪色文字进行纹理保真优化，有效避免了传统方法对模糊字形的误判。

针对视觉质量优化问题，当前研究方法呈现出明显的技术分界：传统算法凭借人工设计特征的高可解释性及低计算复杂度占据基础优势，但其表征能力受限于先验模型构建范式，在动态模糊、弱纹理等复杂成像条件下的泛化性能呈现断崖式衰减；而深度学习方法通过端到端的层次化特征学习机制显著提升了系统的鲁棒性，却伴随着模型参数量激增引发的计算资源消耗与训练数据需求的同步激增。为此，现代分辨率增强技术已突破传统“同质化”超分辨范式，创新性地构建面向工业检测、医学影像、文化遗产修复等垂直领域的自适应增强框架。通过引入时序生成对抗网络建模动态退化过程、构建三维点云高密度重建的物理约束模型以及融合多尺度纹理先验的损失函数，该技术范式成功实现了从低层次像素级复现到高层次语义信息增强的技术跃迁，为不同应用场景下的图像质量提升提供了理论依据与方法支撑。

5. 发展趋势

基于数据驱动模型的现有突破与共性挑战，未来研究将聚焦以下方向：

(1) 提升数据预处理和特征提取的精度

在视觉数据的预处理阶段，对图像进行降噪、增强和归一化等操作至关重要。这些操作能够提高后续模型训练的效率和检测的准确性。特征提取作为视觉数据处理的关键步骤，直接影响到模型的性能。未来的研究需要开发更先进的算法，以提高特征提取的精度和鲁棒性，尤其是在复杂环境下对目标的识别和分类。例如，深度学习模型如卷积神经网络(CNN)和 Transformer 在特征提取方面展现出了强大的能力，未来的研究可以进一步探索这些模型在视觉数据处理中的应用。

(2) 探索基于深度学习的多模态融合技术。

多模态融合技术通过整合来自不同传感器或不同来源的数据，能够提供更全面的图像信息，提高检测的准确性和鲁棒性。深度学习模型，尤其是多模态深度学习模型，为处理和融合多源数据提供了强大的工具。未来的研究可以探索如何利用深度学习模型来处理和融合来自不同模态的数据，以提高视觉数据处理的性能。

(3) 发展小样本学习和模型可解释性

在实际应用中，某些场景下标注数据的获取可能非常昂贵或不可行，这就需要模型能够在少量标注数据上进行有效的学习。小样本学习技术可以帮助模型在数据稀缺的情况下进行学习。同时，模型的可解释性也是视觉数据处理领域的一个重要研究方向，它可以帮助用户理解模型的决策过程，增强对模型的信任。

(4) 推动无监督和自监督学习技术的发展

无监督学习技术可以在没有标注数据的情况下发现数据中的模式和结构，这对于大规模图像数据的处理尤为重要。自监督学习技术通过设计预测任务，使模型能够从未标注的数据中学习有用的特征表示。这些技术的发展将进一步拓宽视觉数据处理技术的应用范围。

(5) 应对实际成像环境的复杂性

实际成像环境的复杂性对视觉数据处理技术提出了挑战，包括光照变化、遮挡、动态场景等问题。未来的研究需要开发更加鲁棒的模型，以应对这些复杂环境的影响。

综上所述，视觉数据处理领域的数据驱动模型正朝着提高预处理和特征提取精度、探索多模态融合技术、发展小样本学习和模型可解释性、推动无监督和自监督学习技术发展以及应对实际成像环境复杂性等方向发展。这些趋势将共同推动视觉数据处理技术的进步，以适应不断增长的实际应用需求。

6. 总结与展望

本文系统综述了视觉数据处理数据驱动模型的发展脉络与技术体系，重点聚焦图像分辨率提升的关键技术路径与应用前景。通过对三维重建、压缩感知、单像素成像及超分辨率等核心技术的剖析，揭示了数据驱动模型从传统机器学习到深度学习(CNN、Transformer、DDPM)的范式演进。研究表明，物理机制与数据驱动的深度融合、多模态协同重建、边缘轻量化部署已成为突破现有技术瓶颈的关键方向。综上，笔者认为视觉数据处理领域下一步的研究重点方向如下。

(1) 多模态融合技术的发展：目前的融合框架多数是基于两种不同来源的图像，未来研究应探索将更多种类的图像数据融合在一起，以期获得更丰富全面的特征信息。这将有效改善当前大部分融合算法提取的特征为单一特征的局限性，提高视觉数据处理的准确性和鲁棒性。

(2) 预处理技术的整合与智能化：随着图像配准、特征提取等预处理技术的快速发展，未来的研究应着力于将这些处理模块与融合模块整合为一套智能的处理系统，实现一站式融合。这将降低对输入图像的要求，扩大多源图像融合技术的应用范围，并提升用户体验。

(3) 评价指标的标准化：融合算法的性能并不总是与融合图像性能完全匹配，现有的评价指标也不完

全等同于图像的主观评价。因此，为了更客观地评价融合效果，未来研究需要制定和规范评价标准，提升评价质量，以更好地指导图像融合技术的发展和应用。

综上所述，视觉数据处理领域的数据驱动模型正面临着新的挑战和机遇，未来的研究将在多模态融合技术、预处理技术的整合与智能化、评价指标的标准化等方面取得新的进展，推动视觉数据处理技术向更高精度、更广应用和更深理解的方向发展。

致　谢

衷心感谢山西省青年科学研究项目“高温环境下炉外磁场差分实现大型矿热炉电极端部位置的在线检测”(202103021223067)所提供的支持。

参考文献

[1] Moulon, P., Monasse, P. and Marlet, R. (2013) Global Fusion of Relative Motions for Robust, Accurate and Scalable Structure from Motion. 2013 *IEEE International Conference on Computer Vision*, Sydney, 1-8 December 2013, 3248-3255. https://doi.org/10.1109/iccv.2013.403

[2] Heller, J., Havlena, M., Jancosek, M., Torii, A. and Pajdla, T. (2015) 3D Reconstruction from Photographs by CMP SfM Web Service. 2015 14*th IAPR International Conference on Machine Vision Applications* (*MVA*), Tokyo, 18-22 May 2015, 30-34. https://doi.org/10.1109/mva.2015.7153126

[3] Schonberger, J.L. and Frahm, J. (2016) Structure-from-Motion Revisited. 2016 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Las Vegas, 27-30 June 2016, 4104-4113. https://doi.org/10.1109/cvpr.2016.445

[4] Cui, H., Gao, X., Shen, S. and Hu, Z. (2017) HSFM: Hybrid Structure-from-Motion. 2017 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Honolulu, 21-26 July 2017, 2393-2402. https://doi.org/10.1109/cvpr.2017.257

[5] Yin, H.Y. and Yu, H.Y. (2020) Incremental SFM 3D Reconstruction Based on Monocular. 2020 13*th International Symposium on Computational Intelligence and Design* (*ISCID*), Hangzhou, 12-13 December 2020, 17-21. https://doi.org/10.1109/iscid51228.2020.00011

[6] Seitz, S.M., Curless, B., Diebel, J., Scharstein, D. and Szeliski, R. (2006) A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. 2006 *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, New York, 17-22 June 2006, 519-528.

[7] Sinha, S.N., Mordohai, P. and Pollefeys, M. (2007) Multi-View Stereo via Graph Cuts on the Dual of an Adaptive Tetrahedral Mesh. 2007 *IEEE* 11*th International Conference on Computer Vision*, Rio de Janeiro, 14-21 October 2007, 1-8. https://doi.org/10.1109/iccv.2007.4408997

[8] Lin, X.B., Wang, J.X. and Lin, C. (2020) Research on 3D Reconstruction in Binocular Stereo Vision Based on Feature Point Matching Method. 2020 *IEEE* 3*rd International Conference on Information Systems and Computer Aided Education* (*ICISCAE*), Dalian, 27-29 September 2020, 551-556. https://doi.org/10.1109/iciscae51034.2020.9236889

[9] Wang, Y.X., Lu, Y.W., Xie, Z.H. and Lu, G.Y. (2021) Deep Unsupervised 3D SfM Face Reconstruction Based on Massive Landmark Bundle Adjustment. *Proceedings of the* 29*th ACM International Conference on Multimedia*, Chengdu, 20-24 October 2021, 1350-1358. https://doi.org/10.1145/3474085.3475689

[10] Lindenberger, P., Sarlin, P., Larsson, V. and Pollefeys, M. (2021) Pixel-Perfect Structure-From-Motion with Featuremetric Refinement. 2021 *IEEE*/*CVF International Conference on Computer Vision* (*ICCV*), Montreal, 10-17 October 2021, 5967-5977. https://doi.org/10.1109/iccv48922.2021.00593

[11] Zhou, L., Zhang, Z., Jiang, H., Sun, H., Bao, H. and Zhang, G. (2021) DP-MVS: Detail Preserving Multi-View Surface Reconstruction of Large-Scale Scenes. *Remote Sensing*, **13**, Article 4569. https://doi.org/10.3390/rs13224569

[12] Eigen D., Puhrsch, C. and Fergus, R. (2014) Depth Map Prediction from a Single Image Using a Multi-Scale Deep Network. *International Conference on Neural Information Processing Systems*, Cambridge, 8-13 December 2014, 2366-2374.

[13] Eigen, D. and Fergus, R. (2015) Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture. 2015 *IEEE International Conference on Computer Vision* (*ICCV*), Santiago, 7-13 December 2015, 2650-2658. https://doi.org/10.1109/iccv.2015.304

[14] Crispell, D. and Bazik, M. (2017) Pix2Face: Direct 3D Face Model Estimation. 2017 *IEEE International Conference on Computer Vision Workshops* (*ICCVW*), Venice, 22-29 October 2017, 2512-2518. https://doi.org/10.1109/iccvw.2017.295

[15] Yao, Y., Luo, Z., Li, S., Fang, T. and Quan, L. (2018) MVSNet: Depth Inference for Unstructured Multi-View Stereo.

In: *Lecture Notes in Computer Science*, Springer, 785-801. https://doi.org/10.1007/978-3-030-01237-3_47

[16] Yao, Y., Luo, Z., Li, S., Shen, T., Fang, T. and Quan, L. (2019) Recurrent MVSNet for High-Resolution Multi-View Stereo Depth Inference. 2019 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), Long Beach, 15-20 June 2019, 5520-5529. https://doi.org/10.1109/cvpr.2019.00567

[17] Chen, R., Han, S., Xu, J. and Su, H. (2019) Point-Based Multi-View Stereo Network. 2019 *IEEE/CVF International Conference on Computer Vision* (*ICCV*), Seoul, 27 October-2 November 2019, 1538-1547. https://doi.org/10.1109/iccv.2019.00162

[18] Zhang, J., Yao, Y., Li, S., Luo, Z. and Fang, T. (2020) Visibility-Aware Multi-View Stereo Network. *Proceedings of the British Machine Vision Conference* 2020, Manchester, 7-10 September 2020, 184-200. https://doi.org/10.5244/c.34.109

[19] Wei, Z., Zhu, Q., Min, C., Chen, Y. and Wang, G. (2021) AA-RMVSNet: Adaptive Aggregation Recurrent Multi-View Stereo Network. 2021 *IEEE/CVF International Conference on Computer Vision* (*ICCV*), Montreal, 10-17 October 2021, 6167-6176. https://doi.org/10.1109/iccv48922.2021.00613

[20] Peng, R., Wang, R., Wang, Z., Lai, Y. and Wang, R. (2022) Rethinking Depth Estimation for Multi-View Stereo: A Unified Representation. 2022 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), New Orleans, 18-24 June 2022, 18-24. https://doi.org/10.1109/cvpr52688.2022.00845

[21] Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R. and Ng, R. (2020) NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In: *Lecture Notes in Computer Science*, Springer, 405-421. https://doi.org/10.1007/978-3-030-58452-8_24

[22] Yen-Chen, L., Florence, P., Barron, J.T., Rodriguez, A., Isola, P. and Lin, T. (2021). iNeRF: Inverting Neural Radiance Fields for Pose Estimation. 2021 *IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS*), Prague, 27 September-1 October 2021, 1323-1330. https://doi.org/10.1109/iros51168.2021.9636708

[23] Xu, Q.G., Xu, Z., Philip, J., Bi, S., Shu, Z., Sunkavalli, K., *et al*. (2022) Point-NeRF: Point-Based Neural Radiance Fields. 2022 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), New Orleans, 18-24 June 2022, 5428-5438. https://doi.org/10.1109/cvpr52688.2022.00536

[24] Xu, L., Xiangli, Y., Peng, S., Pan, X., Zhao, N., Theobalt, C., *et al*. (2023) Grid-Guided Neural Radiance Fields for Large Urban Scenes. 2023 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), Vancouver, 17-24 June 2023, 8296-8306. https://doi.org/10.1109/cvpr52729.2023.00802

[25] Stucker, C. and Schindler, K. (2020) ResDepth: Learned Residual Stereo Reconstruction. 2020 *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (*CVPRW*), Seattle, 14-19 June 2020, 707-716. https://doi.org/10.1109/cvprw50498.2020.00100

[26] Peng, S.D., Zhang, Y.Q., Xu, Y.H., Wang, Q.Q., Shuai, Q., Bao, H.J. and Zhou, X.W. (2021) Neural Body: Implicit Neural Representations with Structured Latent Codes for Novel View Synthesis of Dynamic Humans. 2021 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), Nashville, 20-25 June 2021, 9050-9059. https://doi.org/10.1109/cvpr46437.2021.00894

[27] Huang, Y.H., He, Y., Yuan, Y.J., Lai, Y.K. and Gao, L. (2022) StylizedNeRF: Consistent 3D Scene Stylization as Stylized Nerf via 2D-3D Mutual Learning. 2022 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), New Orleans, 18-24 June 2022, 18321-18331. https://doi.org/10.1109/cvpr52688.2022.01780

[28] Yu, L., Li, X., Fu, C., Cohen-Or, D. and Heng, P. (2018) Pu-Net: Point Cloud Upsampling Network. 2018 *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, 18-23 June 2018, 2790-2799. https://doi.org/10.1109/cvpr.2018.00295

[29] Li, R., Li, X., Fu, C., Cohen-Or, D. and Heng, P. (2019) PU-GAN: A Point Cloud Upsampling Adversarial Network. 2019 *IEEE/CVF International Conference on Computer Vision* (*ICCV*), Seoul, 27 October-2 November 2019, 7202-7211. https://doi.org/10.1109/iccv.2019.00730

[30] He, Y., Tang, D., Zhang, Y., Xue, X. and Fu, Y. (2023) Grad-Pu: Arbitrary-Scale Point Cloud Upsampling via Gradient Descent with Learned Distance Functions. 2023 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), Vancouver, 17-24 June 2023, 5354-5363. https://doi.org/10.1109/cvpr52729.2023.00518

[31] Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R. and Ng, R. (2020) NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In: *Lecture Notes in Computer Science*, Springer, 405-421. https://doi.org/10.1007/978-3-030-58452-8_24

[32] Li, S.Y., Yang, W. and Liao, Q. (2024) PMAFusion: Projection-Based Multi-Modal Alignment for 3D Semantic Occupancy Prediction. 2024 *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (*CVPRW*), Seattle, 17-18 June 2024, 3627-3634. https://doi.org/10.1109/cvprw63382.2024.00366

[33] Jiang, H.W., Huang, Q. and Pavlakos, G. (2024) Real3D: Scaling up Large Reconstruction Models with Real-World Images. https://doi.org/10.48550/arXiv.2406.08479

[34] Yoshida, S., Sun, Z., Yoshizawa, S., Michikawa, T., Noda, S., Micheletto, R., *et al*. (2024). Image Compressed Sensing

Based on Vision-Inspired Importance Maps. 2024 *IEEE International Conference on Imaging Systems and Techniques* (*IST*), Tokyo, 14-16 October 2024, 1-6. https://doi.org/10.1109/ist63414.2024.10759151

[35] He, Y.C., Wang, F., Wang, S.Y. and Chen, B.D. (2017) Diffusion Adaptation Framework for Compressive Sensing Reconstruction. *Signal Processing*, **176**, Article 107660.

[36] Oikonomou, V.P., Nikolopoulos, S. and Kompatsiaris, I. (2019) A Novel Compressive Sensing Scheme under the Variational Bayesian Framework. 2019 27*th European Signal Processing Conference* (*EUSIPCO*), A Coruna, 2-6 September 2019, 1-5. https://doi.org/10.23919/eusipco.2019.8902704

[37] Li, S., Ling, Z. and Zhu, K. (2024) Image Super Resolution by Double Dictionary Learning and Its Application to Tool Wear Monitoring in Micro Milling. *Mechanical Systems and Signal Processing*, **206**, Article 110917. https://doi.org/10.1016/j.ymssp.2023.110917

[38] Beck, A. and Teboulle, M. (2009) A Fast Iterative Shrinkage-Thresholding Algorithm with Application to Wavelet-Based Image Deblurring. 2009 *IEEE International Conference on Acoustics, Speech and Signal Processing*, Taipei, 19-24 April 2009, 693-696. https://doi.org/10.1109/icassp.2009.4959678

[39] Li, C.B. (2010) An Efficient Algorithm for Total Variation Regularization with Applications to the Single Pixel Camera and Compressive Sensing. Master's Thesis, Rice University.

[40] Bian, S. and Zhang, L. (2021) Overview of Match Pursuit Algorithms and Application Comparison in Image Reconstruction. 2021 *IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers* (*IPEC*), Dalian, 14-16 April 2021, 216-221. https://doi.org/10.1109/ipec51340.2021.9421295

[41] Khatib, R., Simon, D. and Elad, M. (2020) Learned Greedy Method (LGM): A Novel Neural Architecture for Sparse Coding and beyond. *Journal of Visual Communication and Image Representation*, **77**, Article 103095. https://doi.org/10.1016/j.jvcir.2021.103095

[42] Li, S., Wang, H., Liu, T., Cui, Z., Chen, J.N. and Xia, Z. (2021) A Fast Barzilai-Borwein Gradient Projection for Sparse Reconstruction Algorithm Based on 3D Modeling: Application to ERT Imaging. *IEEE Access*, **9**, 152913-152922. https://doi.org/10.1109/access.2021.3127695

[43] Yang, Y., Sun, J., Li, H. and Xu, Z. (2020) ADMM-CSNet: A Deep Learning Approach for Image Compressive Sensing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **42**, 521-538. https://doi.org/10.1109/tpami.2018.2883941

[44] Li, X., Ran, J. and Zhou, Z. (2022) An Efficient 3D Radar Imaging Algorithm Based on FISTA. 2022 *IEEE* 9*th International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications* (*MAPE*), Chengdu, 26-29 August 2022, 419-423. https://doi.org/10.1109/mape53743.2022.9935219

[45] Huang, S., Chen, Y. and Qiao, T. (2021) An Extended Reweighted ℓ1 Minimization Algorithm for Image Restoration. *Mathematics*, **9**, Article 3224. https://doi.org/10.3390/math9243224

[46] Edgar, M.P., Gibson, G.M. and Padgett, M.J. (2018) Principles and Prospects for Single-Pixel Imaging. *Nature Photonics*, **13**, 13-20. https://doi.org/10.1038/s41566-018-0300-7

[47] Monin, S., Hahamovich, E. and Rosenthal, A. (2021) Single-Pixel Imaging of Dynamic Objects Using Multi-Frame Motion Estimation. *Scientific Reports*, **11**, Article No. 7712. https://doi.org/10.1038/s41598-021-83810-z

[48] Zhuang, Z., Li, T.H., Wang, H.K., *et al*. (2022) Blind Image Deblurring with Unknown Kernel Size and Substantial Noise. https://doi.org/10.48550/arXiv.2208.09483

[49] Song, H., Nie, X., Su, H., Chen, H., Zhou, Y., Zhao, X., *et al*. (2021) 0.8% Nyquist Computational Ghost Imaging via Non-Experimental Deep Learning. *Optics Communications*, **520**, Article 128450. https://doi.org/10.1016/j.optcom.2022.128450

[50] Zhang, H., Zhao, Q., Xu, W., Wang, Y., Li, F., Liu, S., *et al*. (2024) Optical Single-Channel Color Image Encryption Based on Chaotic Palmprint Phase Masks. *Journal of Optics*, **53**, 3342-3350. https://doi.org/10.1007/s12596-023-01510-5

[51] Liu, S.P., Wu, H., Li, Q., Meng, X. and Yin, Y. (2023) Super-Coding Resolution Single-Pixel Imaging Based on Unpaired Data-Driven Deep Learning. *Optics and Lasers in Engineering*, **170**, Article 107786. https://doi.org/10.1016/j.optlaseng.2023.107786

[52] Liang, J.Y., Cao, J.Z., Sun, G.L., Zhang, K., *et al*. (2021) SwinIR: Image Restoration Using Swin Transformer. 2021 *IEEE/CVF International Conference on Computer Vision Workshops* (*ICCVW*), Montreal, 11-17 October 2021, 1833-1844. https://doi.org/10.1109/iccvw54120.2021.00210

[53] Zhang, K., Liang, J.Y., Van Gool, L. and Timofte, R. (2021) Designing a Practical Degradation Model for Deep Blind Image Super-Resolution. 2021 *IEEE/CVF International Conference on Computer Vision* (*ICCV*), Montreal, 10-17 October 2021, 4771-4780. https://doi.org/10.1109/iccv48922.2021.00475

[54] Salvetti, F., Mazzia, V., Khaliq, A. and Chiaberge, M. (2020) Multi-Image Super Resolution of Remotely Sensed Images

Using Residual Attention Deep Neural Networks. *Remote Sensing*, **12**, Article No. 2207. https://doi.org/10.3390/rs12142207

[55] Song, Q., Xiu, M., Nie, Y., Hu, M. and Liu, C. (2024) CoT-MISR: Marrying Convolution and Transformer for Multi-Image Super-Resolution. *Multimedia Tools and Applications*, **83**, 76891-76903. https://doi.org/10.1007/s11042-024-18591-4

[56] Li, H.A., Zheng, Q.X., Tao, R.L., *et al*. (2023) Review of Image Super-Resolution Based on Deep Learning. *Journal of Graphics*, **44**, 1-15.

[57] 倪劼, 柳青远, 周莉. 利用改进的 Real-ESRGAN 模型进行历史图像超分辨率重建研究[J]. 信息与管理研究, 2025, 10(1): 65-77.

[58] Xie, J. and Cheng, X. (2024) Volleyball Game Tactical Data Prediction System Using Computer Vision Technology. 2024 2*nd International Conference on Mechatronics, IoT and Industrial Informatics* (*ICMIII*), Melbourne, 12-14 June 2024, 588-594. https://doi.org/10.1109/icmiii62623.2024.00116

[59] Nicolson, E., Mohseni, E., Lines, D., Tant, K.M.M., Pierce, G. and MacLeod, C.N. (2024) Towards an In-Process Ultrasonic Phased Array Inspection Method for Narrow-Gap Welds. *NDT & E International*, **144**, Article 103074. https://doi.org/10.1016/j.ndteint.2024.103074

[60] 马钟, 赵歆波, 艾鑫, 张珂. 涡轮叶片X射线图像超分辨率重建技术[J]. CT理论与应用研究(中英文), 2010, 19(1): 41-47.

[61] Sobek, J., Medina Inojosa, J.R., Medina Inojosa, B.J., *et al*. (2023) MedYOLO: A Medical Image Object Detection Framework. https://doi.org/10.48550/arXiv.2312.07729

[62] Guo, Z., Li, X., Huang, H., Guo, N. and Li, Q. (2019) Deep Learning-Based Image Segmentation on Multimodal Medical Imaging. *IEEE Transactions on Radiation and Plasma Medical Sciences*, **3**, 162-169. https://doi.org/10.1109/trpms.2018.2890359

[63] Avram, O., Durmus, B., Rakocz, N., Corradetti, G., An, U., Nittala, M.G., *et al*. (2024) Accurate Prediction of Disease-Risk Factors from Volumetric Medical Scans by a Deep Vision Model Pre-Trained with 2D Scans. *Nature Biomedical Engineering*, **9**, 507-520. https://doi.org/10.1038/s41551-024-01257-9

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(3), 149-160
Published Online June 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.23012

Hans汉斯

CherryUSB原理性分析和应用实践

吕家振

CherryUSB社区，江苏 南京

收稿日期：2025年8月11日；录用日期：2025年10月15日；发布日期：2025年10月27日

摘　要

CherryUSB是一个轻量级、高性能的开源USB主从协议栈，由国内开发者维护，专为资源受限的带USB外设的嵌入式系统设计。近年来，随着带USB外设的嵌入式设备逐渐增多，CherryUSB成为了一个可靠稳定的选择。相比其他USB协议栈，CherryUSB更注重用户阅读体验、驱动全面性、稳定性和高性能，降低了开发者入门的门槛，发挥出了嵌入式设备中USB的优势。本文对CherryUSB主机和从机代码进行原理性的分析，并基于rt-thread artpi2开源硬件平台，进行CherryUSB主从机的应用实践，为嵌入式USB开发提供参考和借鉴。

关键词

USB协议栈，嵌入式系统，CherryUSB项目

CherryUSB Principle Analysis and Application Practice

Jiazhen Lv

CherryUSB Community, Nanjing Jiangsu

Received: August 11, 2025; accepted: October 15, 2025; published: October 27, 2025

Abstract

CherryUSB is a lightweight, high-performance open source USB device and host stack, maintained by domestic developers, designed for resource-constrained embedded systems with USB peripherals. In recent years, with the increasing number of embedded devices with USB peripherals, CherryUSB has become a reliable and stable choice. Compared with other USB protocol stacks, CherryUSB focuses more on user reading experience, driver comprehensiveness, stability, and high performance, lowering the barrier for developers to get started and giving full play to the advantages of USB in embedded devices. This paper analyzes the CherryUSB host and slave code in principle, and conducts

文章引用：吕家振. CherryUSB 原理性分析和应用实践[J]. 嵌入式技术与智能系统, 2025, 2(3): 149-160.
DOI: 10.12677/etis.2025.23012

the application practice of CherryUSB host and slave based on the rt-thread open source hardware platform named art-pi2, and provides a reference for embedded USB development.

Keywords

USB Stack, Embedded System, CherryUSB Project

1. 引言

CherryUSB 项目于 2022 年发布，旨在为资源受限的嵌入式系统提供一套简洁易学且高效的 USB 协议栈[1]。在设计上，CherryUSB 采用树状化编程，所有类驱动和 USBIP 驱动模板化设计，没有复杂的 C 语言语法，方便用户学习和新增驱动。对用户常用的数据收发 API 采用类比法，将复杂 USB 通信类比为 UART + DMA 通信，简化了通信逻辑，方便用户使用。为了突出 USB 的性能，主从驱动直接对接寄存器操作，并且使用零拷贝、硬件 DMA 直通和中断快响应等技术，充分释放 USB 硬件带宽。

自发布以来，CherryUSB 得到了众多开发者和半导体企业的支持，包括博流智能、先楫、平头哥、飞腾、恩智浦、乐鑫、匠心创、博通、嘉楠等。并应用到众多嵌入式设备，包括鼠标、键盘、可视门铃、网络通信、图传、存储、调试、bootrom 等应用。并且 CherryUSB 采用 Apache 2.0 授权，可免费在商用解决方案中使用。

随着热度的提升，有越来越多的开源项目使用 CherryUSB，同样 CherryUSB 也对火热的开源项目进行了支持，包括 daplink，blackmagic，lvgl，lwip，nuttx，zephyr 等，并加入到 rt-thread 国产 OS 项目中，成为 rt-thread 标准 USB 协议栈。

USB 协议栈是一个庞大的体系，如何化繁为简，降低用户开发门槛，理解 USB 的实现原理，是一个难题。本文将对 CherryUSB 的主机和从机代码进行原理性分析，提取关键信息，帮用户理清脉络，并使用 rt-thread artpi2 开源硬件进行移植和主从应用的实践。

2. CherryUSB 从机协议栈概述

CherryUSB 从机协议栈主要包括从机控制器 IP 驱动，USB 设备枚举，USB 驱动加载。其中，设备枚举和驱动加载部分包括 USB 描述符注册，USB 接口驱动注册，USB 端点注册。CherryUSB 设备协议栈支持多种 USB 设备类驱动，包括 CDC、HID、MSC、VIDEO、AUDIO、RNDIS、MTP 等，基本涵盖所有常用类驱动。本章将从 IP 驱动设计切入，分析从机协议栈的整个实现原理。

2.1. 从机控制器 IP 设计

USB 从机控制器 IP 是一种专用集成电路设计模块，用于实现 USB 协议栈的数据链路层功能，使嵌入式系统能够作为 USB 从设备与主机建立通信。负责管理 PHY 交互、数据包处理、端点配置和传输调度等底层操作，同时向上层软件提供标准寄存器定义。而从机控制器 IP 驱动则是根据这些寄存器定义，进行一些配置，使得 USB 设备能够与主机进行通信。目前，在 CherryUSB 中，支持多种从机控制器 IP，商业性的 IP 包括 MUSB、DWC2、FOTG210 等。下面我们以 DWC2 为例，分析该 IP 的设计思路。

1) DWC2 IP 寄存器定义

DWC2 IP 寄存器包含全局寄存器、从机寄存器、主机寄存器三类。从机寄存器中又包含端点寄存器组，包括 IN 端点寄存器组和 OUT 端点寄存器组，各支持最大 16 组。寄存器组的定义如下：

```
typedef struct
{
__IO uint32_t DIEPCTL;          /*!< dev IN Endpoint Control Reg       900h + (ep_num * 20h) + 00h */
uint32_t Reserved04;            /*!< Reserved                          900h + (ep_num * 20h) + 04h */
__IO uint32_t DIEPINT;          /*!< dev IN Endpoint Itr Reg           900h + (ep_num * 20h) + 08h */
uint32_t Reserved0C;            /*!< Reserved                          900h + (ep_num * 20h) + 0Ch */
__IO uint32_t DIEPTSIZ;         /*!< IN Endpoint Txfer Size            900h + (ep_num * 20h) + 10h */
__IO uint32_t DIEPDMA;          /*!< IN Endpoint DMA Address Reg       900h + (ep_num * 20h) + 14h */
__IO uint32_t DTXFSTS;          /*!< IN Endpoint Tx FIFO Status Reg    900h + (ep_num * 20h) + 18h */
uint32_t Reserved18;           /*!< Reserved    900h + (ep_num*20h) + 1Ch-900h + (ep_num * 20h) + 1Ch
*/
} DWC2_INEndpointTypeDef;

typedef struct
{
__IO uint32_t DOEPCTL;          /*!< dev OUT Endpoint Control Reg      B00h + (ep_num * 20h) + 00h */
uint32_t Reserved04;            /*!< Reserved                          B00h + (ep_num * 20h) + 04h */
__IO uint32_t DOEPINT;          /*!< dev OUT Endpoint Itr Reg          B00h + (ep_num * 20h) + 08h */
uint32_t Reserved0C;           /*!< Reserved                          B00h + (ep_num * 20h) + 0Ch */
__IO uint32_t DOEPTSIZ;         /*!< dev OUT Endpoint Txfer Size       B00h + (ep_num * 20h) + 10h */
__IO uint32_t DOEPDMA;          /*!< dev OUT Endpoint DMA Address      B00h + (ep_num * 20h) + 14h */
uint32_t Reserved18[2];         /*!< Reserved B00h + (ep_num * 20h) + 18h - B00h + (ep_num * 20h) +
1Ch */
} DWC2_OUTEndpointTypeDef;
```

从上述寄存器总结如下表 1，从中得出，数据的传输，主要是依靠端点，并配置端点相关的寄存器，下表 1 中 IN 代表发送，OUT 代表接收：

Table 1. DWC2 IP endpoint register description
表 1. DWC2 IP 端点寄存器说明

寄存器	功能
DIEPCTL\DOEPCTL	配置端点属性，类型，最大数据包长度
DIEPTSIZ\DOEPTSIZ	配置端点发送或者接收长度
DIEPDMA\DOEPDMA	配置端点发送或者接收 buffer 地址
DIEPINT\DOEPINT	配置发送或者接收端点中断

2) DWC2 IP 中断

DWC2 的中断标志有很多，这里只列举常用的一些中断，包括：

USB_OTG_GINTSTS_OEPINT
USB_OTG_GINTSTS_IEPINT
USB_OTG_GINTSTS_USBRST
USB_OTG_GINTSTS_USBSUSP
USB_OTG_GINTSTS_WKUINT

USB_OTG_GINTSTS_OEPINT 和 USB_OTG_GINTSTS_IEPINT 代表是端点中断，其余几个表示 USB 的一些状态。在端点中断中，又进一步划分

- USB_OTG_GINTSTS_OEPINT

Table 2. DWC2 IP OUT endpoint interrupt register description
表 2. DWC2 IP OUT 端点中断标志说明

中断标志	功能
USB_OTG_DOEPINT_XFRC	接收完成中断
USB_OTG_DOEPINT_STUP	接收 SETUP 包完成中断

- USB_OTG_GINTSTS_IEPINT

Table 3. DWC2 IP IN endpoint interrupt register description
表 3. DWC2 IP IN 端点中断标志说明

中断标志	功能
USB_OTG_DIEPINT_XFRC	发送完成中断
USB_OTG_DIEPINT_TXFE	发送空中断，用于持续发送数据，直到发送完成

根据表 2 和表 3，可以分析出，从机控制器 IP 的特点包括：多组端点寄存器、发送(IN)完成中断、接收(OUT)完成中断、复位\挂起\恢复等状态中断。同样地，其余从机控制器 IP 也是包含这些特点，根据这些特点，从而能够帮助我们定义从机控制器驱动的 API。

2.2. 从机控制器驱动设计

根据从机控制器 IP 的特点，CherryUSB 中制定了通用标准的从机控制器驱动 API (cherryusb/common/usb_dc.h)。如下表 4 所示。

Table 4. CherryUSB device controller API
表 4. CherryUSB 从机控制器 API

函数	功能
usb_dc_init	初始化从机控制器
usb_dc_deinit	反初始化从机控制器
usbd_ep_open	配置端点相关属性

续表

usbd_ep_close	关闭端点
usbd_ep_start_write	端点启动端点发送
usbd_ep_start_read	端点启动端点接收
usbd_event_ep0_setup_complete_handler	setup 包完成中断处理
usbd_event_ep_in_complete_handler	端点发送完成中断处理
usbd_event_ep_out_complete_handler	端点接收完成中断处理

API 和寄存器中大量提到了 USB 的一个名词：端点(endpoint) [2]。端点是 USB 设备内部的基本通信结构单元，本质上是一块缓冲区存储器，用作数据传输的源或目的地，并通过管道(pipe)进行传输。端点具有方向性，包括 IN 和 OUT 方向；端点支持一种传输方式，包括控制传输、批量传输、中断传输、同步传输[3]；端点传输一个包时有最大数据包长度限制，称为 MPS。这里，我们采用类比法，将端点比作 DMA 通道，端点的发送与接收看作是外设发送和接收，端点的传输方式看作是 UART\SPI\I2C 传输等等，端点的完成中断看作是 DMA 通道完成中断。当我们采用类比法以后，可以更快地理解端点的含义和作用，更快的理解从机控制器 IP 和驱动的设计思想。

2.3. 从机协议栈设计

在 USB 中，另一个重要的概念叫做 setup 包[4]，它是主机和从机枚举所定义的一个格式，如图 1 所示。主机和从机通过端点 0 进行控制传输，并发送 setup 包给到从机，从机会执行到 usbd_event_ep0_setup_complete_handler 函数，该函数则是用于处理获取的 setup 包，并按照不同的 bRequest 执行不同的处理，最终完成 USB 的枚举以及接口驱动的加载工作。

Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bitmap	Characteristics of request: D7: Data transfer direction 0 = Host-to-device 1 = Device-to-host D6...5: Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4...0: Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4...31 = Reserved
1	*bRequest*	1	Value	Specific request (refer to Table 9-3)
2	*wValue*	2	Value	Word-sized field that varies according to request
4	*wIndex*	2	Index or Offset	Word-sized field that varies according to request; typically used to pass an index or offset
6	*wLength*	2	Count	Number of bytes to transfer if there is a Data stage

Figure 1. Format of setup packet
图 1. setup 包格式

下图 2 完整描述了 CherryUSB 从机协议栈的执行过程和调用关系，和从机控制器 IP 驱动设计紧密结合。

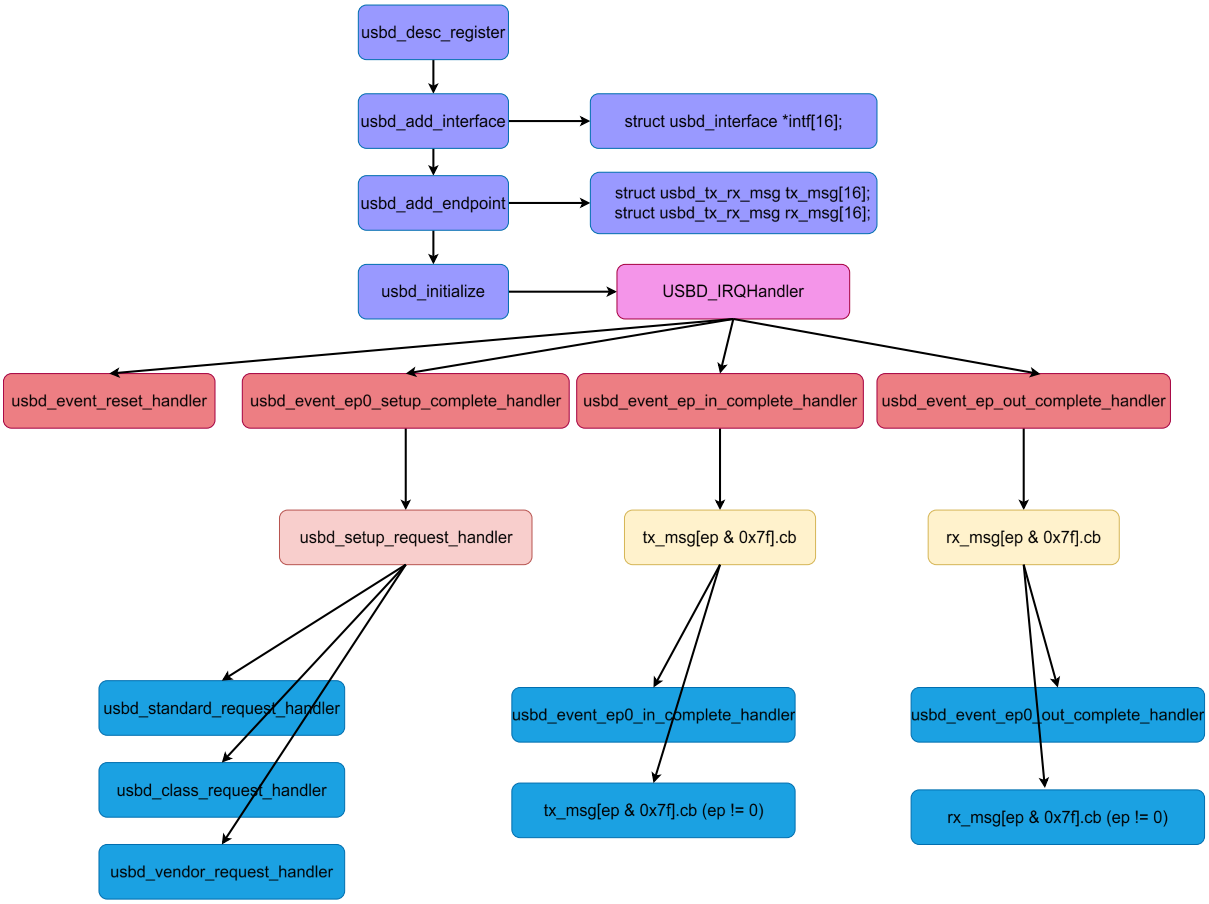

Figure 2. CherryUSB device stack framework
图 2. CherryUSB 设备协议栈框图

3. CherryUSB 主机协议栈概述

CherryUSB 主机协议栈主要包括主机控制器 IP 驱动，USB 设备枚举和驱动加载。相比于从机的一对一的通信方式，主机更具复杂性，由于是一对多的方式，同时需要支持多个 USB 设备，因此，在 CherryUSB 中，采用了 RTOS 的接管，如果使用 bare mental 的方式管理，增加例如死等，状态机的代码，会让用户在阅读和使用上变得非常麻烦，并且也让代码的健壮性变得不可控。CherryUSB 主机协议栈支持多种 USB 设备类驱动，包括 CDC、HID、MSC、VIDEO、AUDIO、RNDIS、HUB 等，可以支持 1 托多个 USB 设备，每个设备还可以是复合设备(支持多种 USB 类接口的设备)。本章同从机一样，从主机控制器 IP 设计入手进行分析。

3.1. 主机控制器 IP 设计

USB 主机控制器 IP 是一个为嵌入式系统提供实现 USB 主机功能的硬件模块，主要负责总线仲裁，数据传输调度，USB 协议的处理，并向上层软件提供标准寄存器定义。而主机控制器 IP 驱动则是根据这些寄存器定义，进行配置并能够与 USB 设备进行通信。CherryUSB 支持多个主机控制器 IP，商业性的 IP

包括 DWC2、MUSB、OHCI、EHCI、XHCI 等。其中 MUSB 和 DWC2 部分模式采用寄存器配置，OHCI/EHCI/XHCI 和部分 DWC2 部分模式采用描述符链表配置，前者更多是把功能交给软件做，而后者则是将复杂功能交给硬件做，降低了软件的负载率，但是同时也增加了软件的编写难度。OHCI/EHCI/XHCI 是 intel 制定的一套主机控制器标准，目前也是市面上最流行也是最通用的，也是最推荐使用的。由于主机控制器设计较为复杂，本章只简单介绍中断设计相关。如下表 5 所示。

Table 5. Interrupt of USB host controller
表 5. USB 主机控制器中断种类

IP	中断标志	功能
DWC2	USB_OTG_GINTSTS_HCINT	中断完成/错误中断
MUSB	MUSB_TXIS/MUSB_RXIS	所有 pipe 中断完成/错误中断
OHCI	XHCI_USBSTS	所有 pipe 中断完成/错误中断
EHCI	EHCI_USBSTS_INT/EHCI_USBSTS_ERR	所有 pipe 中断完成/错误中断
XHCI	OHCI_INT_WDH	所有 pipe 中断完成/错误中断

从上述寄存器功能可以看出，无论主机控制器如何设计，都是具备相似点的，无非只是软件编写方式的不同。在这里 pipe 和从机的端点是对应的，一个端点对应一个 pipe，自然一个 pipe 对应一个端点，但是由于主机是一对多的方式，因此存在多个相同的端点，但是属于不同的 USB 设备，因此，在主机中，统称为 pipe。借助中断，我们可以定义出主机控制器 IP 的驱动。

3.2. 主机控制器驱动设计

主机控制器 IP 驱动，CherryUSB 参考了 linux 的 urb 设计[5]，也是采用 urb (usb request block)的方式，urb 内容如下：

```
struct usbh_urb {
    usb_slist_t list;
    void *hcpriv;
    struct usbh_hubport *hport;
    struct usb_endpoint_descriptor *ep;
    uint8_t data_toggle;
    uint32_t interval;
    struct usb_setup_packet *setup;
    uint8_t *transfer_buffer;
    uint32_t transfer_buffer_length;
    int transfer_flags;
    uint32_t actual_length;
    uint32_t timeout;
    int errorcode;
    uint32_t num_of_iso_packets;
    uint32_t start_frame;
    usbh_complete_callback_t complete;
```

```
    void *arg;
#if defined(__ICCARM__) || defined(__ICCRISCV__) || defined(__ICCRX__)
    struct usbh_iso_frame_packet *iso_packet;
#else
    struct usbh_iso_frame_packet iso_packet[0];
#endif
};
```

- hport 对应一个 USB 设备，当前 urb 将会在哪个 USB 设备上使用
- ep 对应一个 USB 设备上的一个端点，当前 urb 会和该 USB 设备的哪个端点通信
- setup 表示控制传输对应的 setup 包
- transfer_buffer 则表示传输的地址
- transfer_buffer_length 表示传输的长度
- actual_length 表示最终发送或者接收的实际长度

虽然 urb 看上去内容很多，但是实际上，和从机协议栈一样，最终都是端点通信，并且都会触发端点的完成中断，只不过在主机叫做 pipe 完成中断，同样也是可以类比成 UART + DMA 的形式，将 pipe 比作 DMA 通道，传输比作一个外设，比如 UART/SPI/I2C，完成中断比作是 DMA 通道完成中断。

3.3. 主机协议栈设计

主机协议栈主要是对接入的 USB 设备进行枚举，并且解析描述符，再根据描述符找到对应的 USB 类驱动，并初始化类驱动，最后提供用户可用的 API 供调用，对拔出的设备，进行资源的释放。关于主机协议栈整体调用，如图 3 所示。

在主机中，有一个叫做 hub 的概念，USB hub (USB 集线器)是一种允许多个 USB 设备连接到一个单个 USB 端口的设备。它基本上是 USB 接口的分线器或扩展器。每个 hub 上的 USB 接口，称为 hubport，每个 hubport 可以接入一个 USB 设备，因此 USB 设备的枚举就是通过 hub 上的 hubport 进行。而 hub 又分 roothub 和 external hub，roothub 代表主机控制器本身，external hub 自身是一个 USB 设备，具备连接多个 USB 设备的功能，如图 4 所示。因此在 CherryUSB 中会创建一个 hub 守护线程，作用就是探测 hub 上所有 USB 接口是否有插入和拔出事件，如果是插入事件，则进行枚举和驱动加载，如果是拔出事件则进行资源释放。如果接入的是 hub 设备，则通过 hub 通信继续检测插拔事件，最终都会通过统一的一个 hub 守护线程去操作 hub 上的所有 USB 端口。

4. 基于 CherryUSB 的应用开发

本章中，我们将基于 RT-Thread ART-PI2 开源硬件，进行 CherryUSB 的主机和从机移植和应用开发。

4.1. 使用从机枚举虚拟串口

首先根据 CherryUSB 官方文档(https://cherryusb.cherry-embedded.org/quick_start/transplant#usb-device)完成从机的基础移植，主要包含非 USBIP 相关的内容，包括 USB 引脚，USB 时钟，USB 中断等，这些不属于 USBIP 驱动中的内容。由于我们使用 RT-Thread ART-PI2，可以直接使用 ART-PI2 官方仓库(https://github.com/RT-Thread-Studio/sdk-bsp-stm32h7r-realthread-artpi2)，并选择 art_pi2_cherryusb_usbdev_cdc_acm 工程编译即可。具体则是在 env 环境中使用 menuconfig 勾选 CherryUSB 模块，选择 USBIP 和 cdc acm 类和 demo 并保存，如图 5 所示。

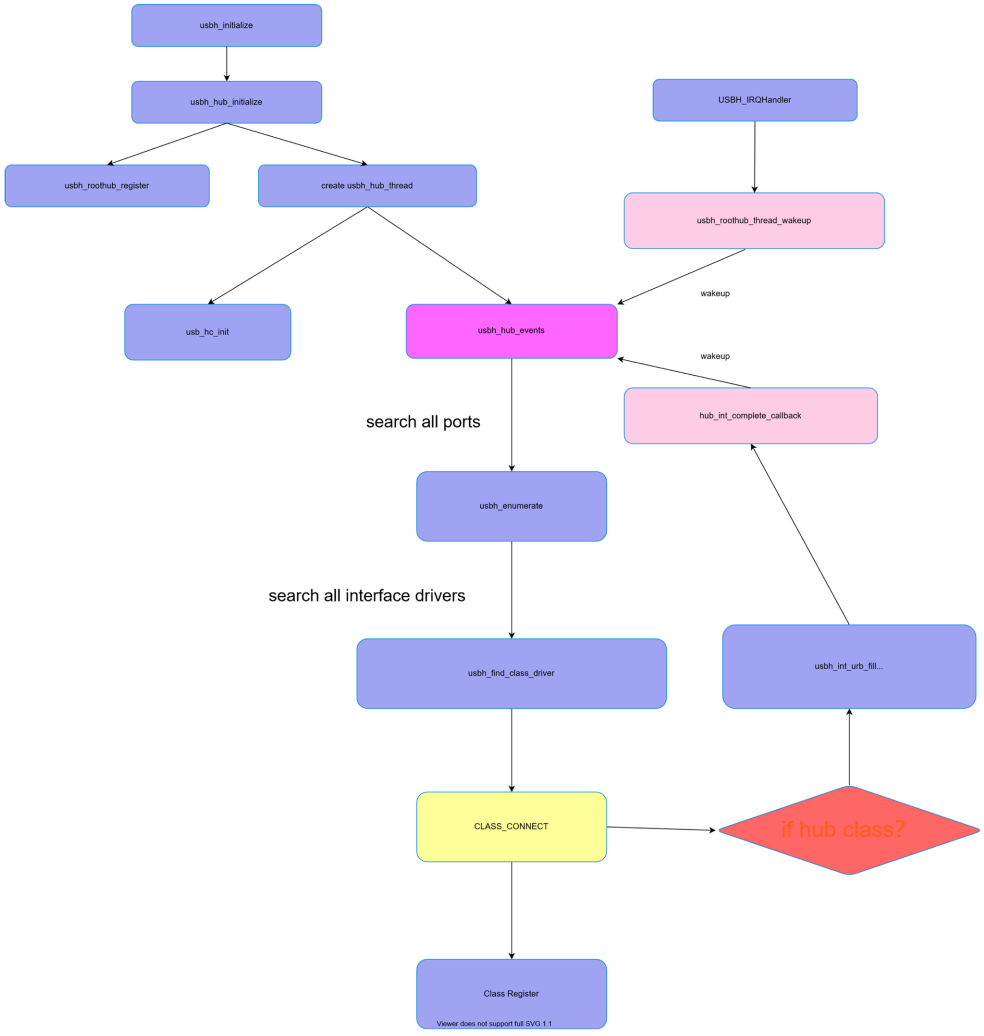

Figure 3. CherryUSB host stack framework
图 3. CherryUSB 主机协议栈框图

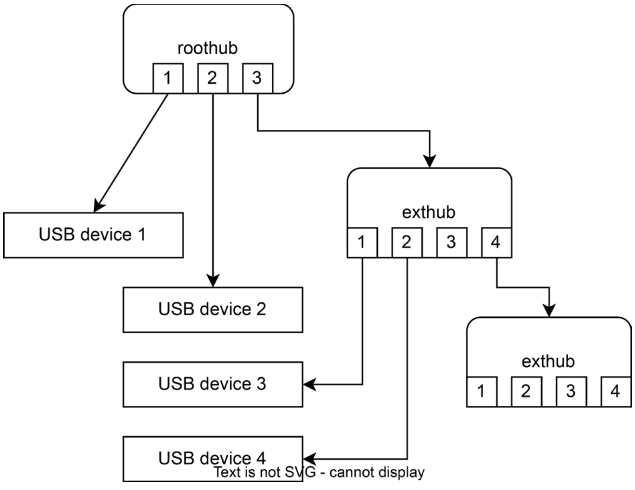

Figure 4. USB hub topological structure
图 4. USB hub 拓扑结构

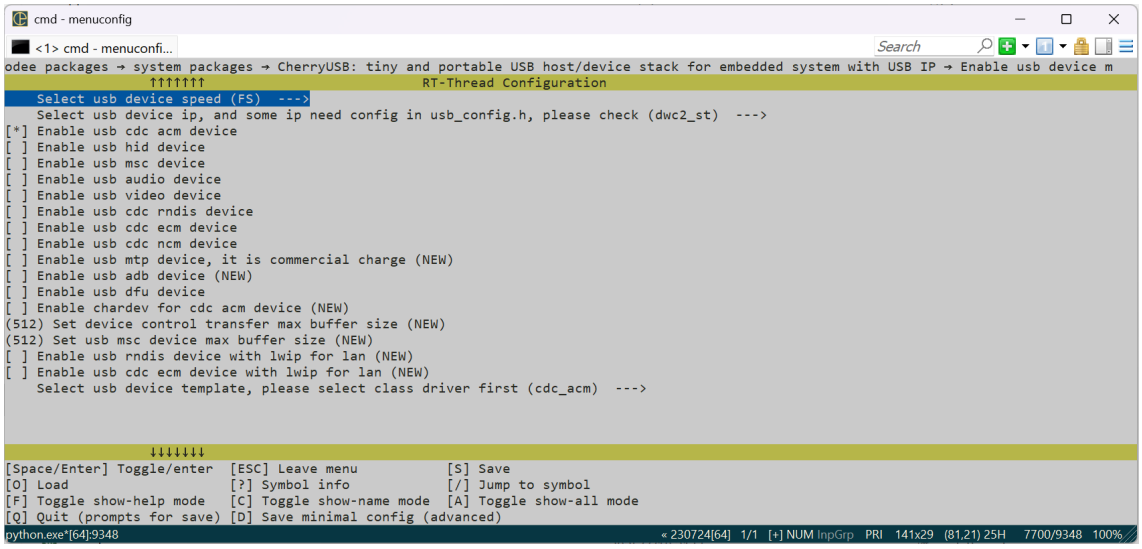

Figure 5. CherryUSB device menuconfig
图 5. CherryUSB 从机 menuconfig 配置

然后执行 scons --target=mdk5，并在 main 函数中调用 cdc_acm_init 初始化 cdc acm 设备，然后编译构建即可。最终将会在电脑上显示一个 USB COM 口，并可以进行数据的通信。如图 6 所示。

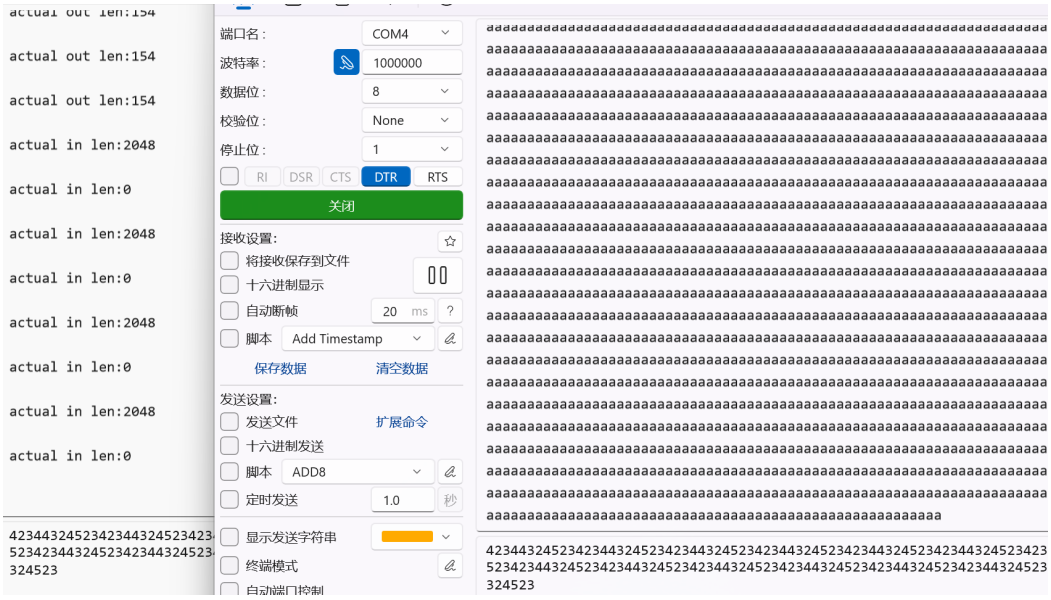

Figure 6. CDC ACM read write test
图 6. CDC ACM 读写测试

4.2. 使用主机枚举 U 盘

主机的移植可以参考 CherryUSB 官方文档 (https://cherryusb.cherry-embedded.org/quick_start/transplant#usb-host)，同样在 ART-PI2 官方仓库中也包含主机的例程，选择 art_pi2_cherryusb_usbhost 工程[6]，并使用 menuconfig 勾选 USBIP 和 MSC 类设备并保存，如图 7 所示。CherryUSB 将会对接到 RT-Thread 的 DFS (虚拟文件系统)组件中。

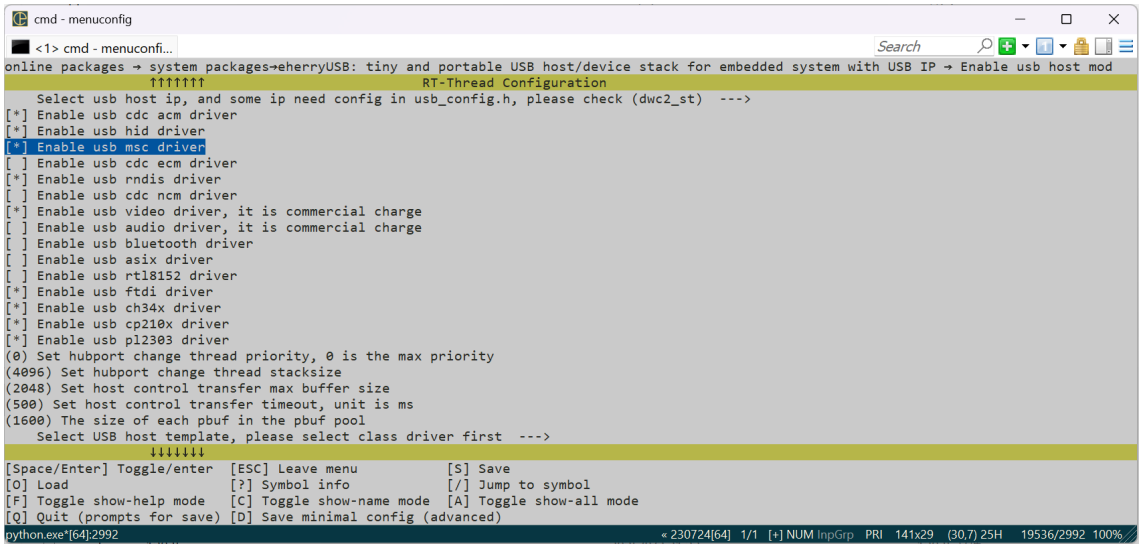

Figure 7. CherryUSB host menuconfig
图 7. CherryUSB 主机 menuconfig 配置

然后执行 scons--target=mdk5，并在 main 函数中使用 usbh_initialize 初始化主机协议栈初始化，然后编译构建即可。

当我们插入一个 U 盘时，协议栈会识别并进行枚举，加载 MSC 驱动，最终注册到 DFS 组件中，可以使用‘ls’，‘cat’，‘echo’相关 API 进行访问。如图 8 所示。

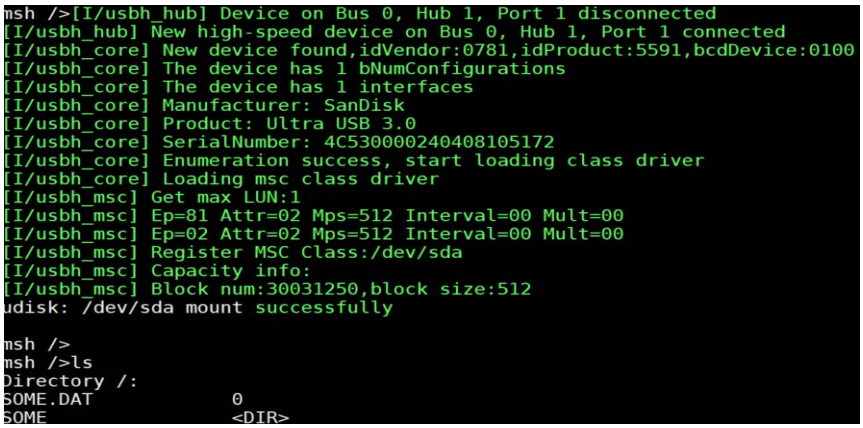

Figure 8. CherryUSB host flash drive test
图 8. CherryUSB 主机接入 U 盘测试

5. 结语

CherryUSB 的设计是为了让用户更好地理解和使用 USB，将复杂的功能采用简易代码和类比等形式简化，并实现了众多 USB 类设备，统一了 USBIP 驱动，降低了用户移植到新嵌入式系统中的难度，也方便了嵌入式开发者进行应用开发或者是 IP 验证。CherryUSB 希望通过这些来吸引更多的开发者和企业用户。希望通过本文，读者对 CherryUSB 的原理有个初步的了解，不必拘泥于复杂的 USB 概念，而是从 USBIP 设计入手去发现规律，总结规律，从而形成一个大体的框架，最终对 USB 的整体会有一个很深入的理解。

CherryUSB 近年来得到了很多开发者和企业的支持，在很多产品当中都有 CherryUSB 的身影，项目也是趋于高度稳定，期待 CherryUSB 在未来能够支持更多的硬件支持，支持更多的开源项目，成为一个主流标准的 USB 协议栈[7]。

参考文献

[1] CherryUSB 官方文档[EB/OL]. https://cherryusb.cherry-embedded.org, 2025-08-04.

[2] The Micriμm USB Team. 嵌入式协议栈 uc/USB-Device [M]. 何小庆, 译. 北京: 航空航天大学出版社, 2015.

[3] 刘荣. 圈圈教你玩 USB [M]. 第 3 版. 北京: 航空航天大学出版社, 2022.

[4] USB Implementers Forum (2000) Universal Serial Bus Specification Revision 2.0. https://www.usb.org/document-library/usb-20-specification

[5] (2000) Linux Project USB Request Block. https://docs.linuxkernel.org.cn/driver-api/usb/URB.html

[6] (2025) RT-Thread Artpi2 Project. https://github.com/RT-Thread-Studio/sdk-bsp-stm32h7r-realthread-artpi2

[7] 王小强. 多种嵌入式平台通用 USB2.0 协议栈的研究与设计[D]: [硕士学位论文]. 成都: 电子科技大学, 2009.

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(3), 161-168
Published Online June 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.23013

Hans汉斯

嵌入式虚拟化技术探索及实践

张云飞[1]，吴春光[2]

[1]麒麟软件有限公司，天津
[2]麒麟软件有限公司，北京

收稿日期：2025年8月12日；录用日期：2025年10月15日；发布日期：2025年10月27日

摘　要

本文介绍了关于嵌入式虚拟化技术的主要类型和特点。针对虚拟化技术在嵌入式领域中应用的主要问题，提出了降低虚拟化损耗和提升系统实时性的方法。并针对嵌入式设备的需求，给出虚拟化下动态调频的技术方案，用于提升系统性能的同时，降低设备功耗。最后列举了两个关于不同虚拟化类型在工业场景下的实际应用。

关键词

虚拟化，实时性，设备直通

Exploration and Practice of Embedded Virtualization Technology

Yunfei Zhang[1], Chunguang Wu[2]

[1]KylinSoft, Tianjin
[2]KylinSoft, Beijing

Received: August 12, 2025; accepted: October 15, 2025; published: October 27, 2025

Abstract

This article introduces the main types and characteristics of embedded virtualization technology. In view of the main problems in the application of virtualization technology in the embedded field, methods to reduce virtualization losses and improve system real-time performance are proposed. In response to the needs of embedded devices, a technical solution for dynamic frequency modulation under virtualization is provided to improve system performance and reduce device power consumption. Finally, two practical applications of different virtualization types in industrial scenarios are listed.

文章引用: 张云飞, 吴春光. 嵌入式虚拟化技术探索及实践[J]. 嵌入式技术与智能系统, 2025, 2(3): 161-168.
DOI: 10.12677/etis.2025.23013

Keywords

Virtualization, Real-Time, Passthrough

1. 引言

虚拟化技术应用较多的是通用的云场景主要以数据中心服务器为依托，虚拟出较多的硬件资源，包括 CPU、显卡、硬盘等资源，帮助企业提高资源利用率、降低成本、提高灵活性和可靠性，从而推动企业更好地适应数字化时代的挑战。这种场景下，对虚拟化的要求主要是灵活应用、降本增效，而不过分关注虚拟化的实时性能。随着物联网、智能制造、自动驾驶等业务的增长，对虚拟化也提出了不同的要求，我们称之为实时虚拟化。在该场景下，我们需要一定的虚拟化能力，但也同样看重实时性能。虚拟化性能或者称为虚拟化损耗是衡量一项虚拟化技术的重要指标。虚拟化是在硬件层之上进行了封装，相比直接基于物理机，必然会损失一部分性能。这对于本就资源贫乏的嵌入式设备来说可能更不容易接受。另外对于一些实时性要求较高的场景，如工控领域，虚拟化损耗将比普通的云场景虚拟化显得尤为重要。

2. 调度机制

在实时性要求较高的情况下，一般都会在虚拟化层之上运行一个实时操作系统作为客户机，通过这个实时操作系统 + 强实时虚拟化来处理实时任务。当一台物理机上并行运行多个虚拟机时，物理机资源的使用率越高，虚拟机性能下降得越剧烈。这就要求实时操作系统 + 强实时虚拟化架构下的实时性指标进一步提高。在这种架构下，Guest OS 的调度器和 Hypervisor 自身的调度器共同构成了两级调度，这里称 Guest OS 对任务的调度为第二级调度，称 Hypervisor 对 VCPU 的调度为第一级调度[1]。两级调度模型抽象了 Guest OS 内部任务在 CPU 上的执行行为，这非常适合用于分析运行于 Hypervisor 上的 RTOS 是否仍然满足实时任务的实时需求。参与调度的主体主要包括客户 OS 中运行的任务 Task、客户 OS 的调度器、VCPU 以及 Hypervisor 的调度器。在两级调度框架下，Guest OS 按照自身的调度算法选取任务使之在 VCPU 上执行，然后 Hypervisor 按照 VCPU 调度算法选取 VCPU 使之能在 CPU 上执行。本文的研究重点是第一级调度，在目前的主流虚拟化软件中主要可以分为静态隔离和动态调度两种方式。

2.1. 静态隔离和动态调度

静态隔离是在虚拟机启动前，就已为其分配固定的物理资源(CPU 核心、内存、PCI 设备等)，上线后不随负载变化动态调整。各虚拟机间资源完全独占，互不干扰，适合对性能和实时性有严格保证的场景。静态隔离的性能、安全可靠性较好，所以工业场景很多情况下更适合静态隔离(成本不敏感，性能优先，持续性任务)。但这种方案的缺点是灵活性、可配置性较差，硬件共享实现困难，导致整个系统的资源利用率差。

动态调度可以根据各虚拟机/容器的实时负载，动态地分配或回收 CPU、内存、I/O 带宽等资源，可在同一物理机上运行超过物理资源总量(overcommit)，提高资源利用率，支持虚拟机的热迁移(live migration)、自动伸缩(autoscaling)等高级功能。例如在 AI 应用场景中，通过虚拟化技术，可以动态调整虚拟机的资源配置(如 CPU、内存)，以适应不同 AI 任务的计算需求，实现资源的最优利用。适合成本敏感，间

歇性任务。

2.2. 动态和静态相结合方案

动态和静态相结合方案首先是基于一个动态调度的框架。本方案将 Hypervisor 上的客户机分为两种类型，即实时客户机和非实时客户机，如图 1 所示。其中 Guest1 到 GuestN 是非实时客户机，这个类型的虚拟机更强调资源虚拟化和 VCPU 调度，满足业务场景对虚拟化的要求。RT-guest 是实时虚拟机，这个类型旨在满足硬实时的要求，去处理实时任务。RT-guest 中使用 bind 调度，使 CPU 对应的 VCPU 队列中只有一个 VCPU，这样对于 RT-guest 来说就不存在 VCPU 调度延时。相应的中断处理也可以直接分发到目标 VCPU，进一步降低了因调度导致的中断延时。在这种情况下，客户机中的物理时钟等于虚拟时钟，保证了 RT-guest 中虚拟机高精度时间应用程序的实时性能。

创建两个 CPU 池，目的是将实时客户机和非实时客户机的物理 CPU 隔离开，使两个域不会互相影响。这里假设系统一共有 n + m 个 CPU，非实时客户机使用 CPU 池 Pool-0，包括 CPU0-CPUn-1，实时客户机使用 CPU 池 rtpool，包含 CPUn-CPUn + m。Pool-0 用于客户的多虚拟化业务，可以创建多个虚拟机，这些虚拟机共用一个 CPU 池和一套硬件如(Network Interface、UART、DSIK 等)。rtpool 只用于客户实时场景，并且只能给一个实时客户机使用。在实际情况中 n 远大于 m，甚至某些场景只需要一个 CPU (m = 0)去完成实时控制业务。设置 vcpu 硬件亲和性将 rtpool 中的 CPU 指定 VCPU。

方案的关键在于在一个基于动态调度的框架下如何实现静态隔离的功能。因为在动态调度的框架下 VCPU 的调度是基于时钟的调度器去实现的。以 arm 为例，Hypervisor 通过 arm 的 generic timer 产生时钟中断，同时也为系统提供系统时钟。虚拟化的时钟可以简单分为物理时钟和虚拟时钟，正常情况下虚拟时钟等于物理时钟加偏移，但是在 bind 调度中时钟偏移等于 0，所以这时候物理时钟就是虚拟时钟。bind 调度首先关闭调度器的调度定时器，这样就不会周期性地频繁进入调度器，保证了 RTOS 系统不会被打断。每次 generic timer 产生中断以后不再不需要去维护定时器，而是直接将时钟中断注入到 RT-guest 虚拟机中，为 RTOS 系统提供调度时钟。在 bind 调度在初始化时关掉调度定时器，同时完成 VCPU 队列初始化。系统正常运行以后，CPU 将直接运行队列中的 VCPU 实体，而且没有定时器的打断，CPU 将不再由客户机陷入 Hypervisor 的调度器，大大降低了 Hypervisor 的虚拟化损耗。

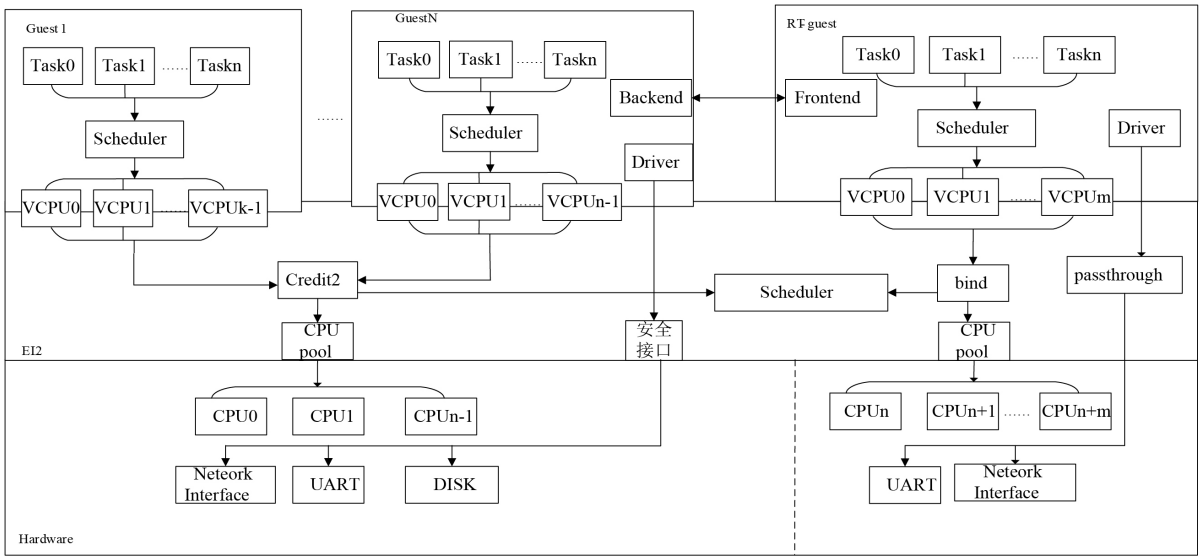

Figure 1. Diagram of the dynamic scheduling + static isolation
图 1. 动态调度 + 静态隔离图

3. ARM 处理器架构下设备直通

设备直通(Passthrough)是一项重要的虚拟化技术，它允许虚拟机(VM)直接访问物理主机上的硬件设备，而不是通过虚拟化层(Hypervisor)。这可以提供更接近物理机性能的虚拟机性能，并在一些场景下提高了虚拟化环境的灵活性。通过虚拟化管理工具或者命令行工具，管理员可以将物理设备分配给特定的虚拟机。这将设备从主机操作系统的控制下移交给虚拟机。在虚拟机生命周期结束或者管理员的操作下，设备可以被释放回主机操作系统控制。这通常涉及对设备的重置和重新分配。设备直通的一个主要优势是它可以提供接近本地性能的虚拟机性能，因为虚拟机可以直接访问硬件资源，而无需通过虚拟化层的中介。

在支持设备直通的系统中，通常需要启用 IOMMU [2]。IOMMU 是一种硬件设备，其主要功能是管理和映射设备的直接内存访问(DMA)。DMA 允许外部设备(如网络适配器、图形卡等)直接访问系统内存，以提高数据传输速度。IOMMU 的作用就是在系统内存和外部设备之间创建一个映射，以提供更好的内存管理和安全性，它为虚拟机提供一个虚拟化的地址空间，隔离虚拟机和其他虚拟机对设备的访问。软件基本都是基于 IOMMU 硬件来实现设备直通功能的。但是 IOMMU 作为一种高级硬件，并不是在所有平台上都支持。尤其是以硬件资源紧缺的嵌入式平台为甚，嵌入式系统中硬件资源有限的情况是相当普遍的，这可能包括有限的处理能力、内存容量、存储空间和其他资源。这种资源的紧张性可能由于多种原因引起，包括功耗要求、成本约束、尺寸限制以及应用场景。基于这种情况，使得虚拟化的设备直通技术在嵌入式场景下难以应用落地。而嵌入式场景又是一个对性能和实时性要求极高的领域，所以这种场景下迫切的需要一种无需硬件 IOMMU 支持的设备直通虚拟化技术。

在虚拟化的传统方案(IOMMU 支持)如图 2 所示 IOMMU 场景下，CPU 的 MMU 地址映射分为两个阶段 Stage1 和 Stage2。Stage1 基于操作系统页表将 CPU 的虚拟地址 VA，转换为中间物理地址 IPA，Stage2 基于 Hypervisor 的页表将 IPA 转换为真正的物理地址 PA [3]。Hypervisor 本身是一套全虚拟化平台，操作系统认为自己是运行在一个真实的硬件中，它并不能感知到 IPA 的存在。对于操作系统来说，它认为的 PA 其实是虚拟化环境下的 IPA。CPU 通过 DMA engine (DMA 控制器驱动)去启动 DMA 去完成一次数据传输。以数据传输方向为内存到设备(网卡、显卡等设备)为例，CPU 将物理内存地址 PA (上文中提高，其实是虚拟化下的 IPA)、设备物理地址、数据大小等内容告诉 DMA，DMA 按照这些配置去完成一次数据传输。当 DMA 进行内存访问时，IOMMU 中已经被 Hypervisor 配置了同样的页表，也就是说 IOMMU 的 IPA- > PA 与 MMU Stage2 的 IPA- > PA 相同，所以 DMA 就可以访问到正确的物理地址。

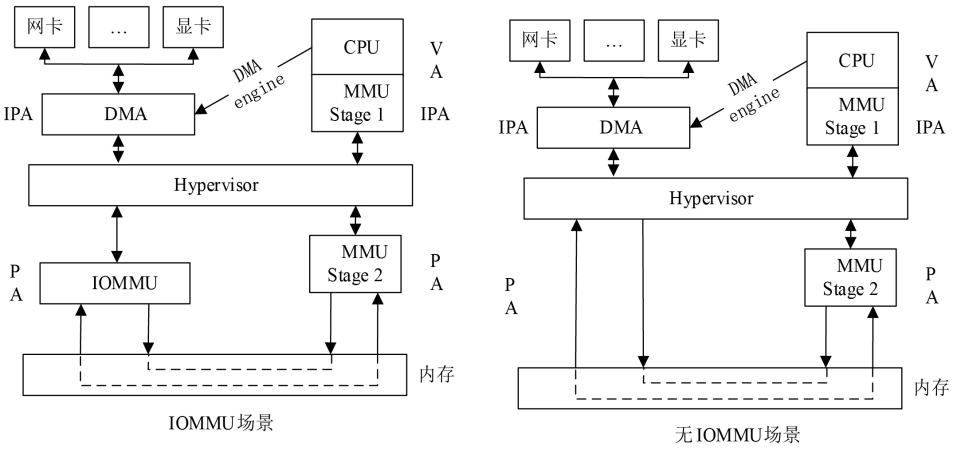

Figure 2. Diagram of memory pass-through under virtualization
图 2. 虚拟化下内存直通原理图

第一步是在创建虚拟机之前，首先通过 Hypervisor 的内存管理器为虚拟机分配静态连续内存。这样指示 Hypervisor 直接从其堆空间中分配一段静态内存，而不是在虚拟机启动以后通过写时复制的手段为虚拟机动态扩充运行内存。在这段静态连续内存上我们可以直接分配 DMA 内存缓冲区。第二步，一般动态调度类型的 Hypervisor 问题在于它的内存是动态分配的，客户机运行的地址空间一般是固定的，所以正常来说 IPA 不等于 PA。DMA 的内存访问必须将 IPA 转换为 PA 才能找到正确的数据。本文首先在客户机的数据结构中找到 Stage2 的页表。同时我们已经在第一步中得到了静态连续内存的起始物理页帧号 mfn，大小为 2^order 个页面。这里修改 gfn (客户机页帧号)等于 mfn，循环建立 mfn 到 gfn 的页表项，最终使虚拟化下的 IPA 等于 PA。如图 2 所示无 IOMMU 场景，DMA 直接使用 DMA engine 配置的内存源地址 IPA 在内存中寻址。但这时经过映射以后 IPA 已经等于 PA，所以 DMA 就可以使用 PA 找到内存中的正确数据。

4. 虚拟化下调频功能

CPU 调频允许动态调整 CPU 频率，从而在性能和节能方面都带来了好处。调频技术对于嵌入式设备来说主要为了解决以下关键问题：一是性能提升，CPU 动态调频使得在必要时可以提高 CPU 频率。这对于“重负载”用例特别有帮助，其中应用程序需要更多的处理能力。通过使用更高的频率，任务可以更快地完成，从而提升整个系统的性能。在加快引导过程，增加 CPU 频率可以加快引导速度，使系统更快地变为可操作状态。这在需要快速启动时间的设备中尤为有益。第二是节能，CPU 动态调频的一个主要优势是能够在系统负载较轻时降低 CPU 频率。在嵌入式设备的嵌入式领域尤为关键。通过降低频率，设备的能耗可以减少。在某些情况下，CPU 动态调频可以帮助防止 CPU 过热。性能优化和功耗效率的结合，使得 CPU 动态调频成为嵌入式系统中的一个有价值的功能。它与资源受限的设备要求相吻合，在性能和能耗之间取得合适的平衡至关重要。

但是在设备引入虚拟化技术以后，CPU 调频就变得大不相同[4]。如图 3 所示，调频的依据是 CPU 的负载计算，也就是说 CPU 负载大，则需要较高的频率以提高 CPU 的算力和实时性。当 CPU 的负载较小时需要降低 CPU 的频率，以减少能量的消耗。开启虚拟化以后，会在系统之上虚拟出多个客户机，不同的客户机分处不同的域，并且相互隔离。对于特权客户机来说，它拥有整个系统的硬件资源权限，这其中就包括 CPU 调频驱动及相应的硬件权限。但特权客户机操作系统只能计算自身虚拟机的负载情况，并不能感知其它虚拟机的负载。如果让特权客户机直接去进行 CPU 动态调频的话，就不能做出正确的决策，影响其它客户机的运行。另外还有一种思路是在 Hypervisor 中实现 CPU 动态调频功能。因为 Hypervisor 了解每个客户机和物理 CPU 的运行情况，所以它来做 CPU 调频的决策是比较正确的。但问题主要是像 TYPE-I 型 Hypervisor 基本上都具有轻量化的特点。它是处在硬件和操作系统之间薄薄的一层。而如果 Hypervisor 要去完成整个 CPU 调频功能的话，需要集成大量的 CPU 调频驱动。因为不同芯片的 CPU 调频驱动千差万别，而为了适配不同的硬件，其代码量是相当巨大，而且考虑到后续的维护，也有很多的工作量。将大量的驱动代码移植到 Hypervisor 中，显然不是 TYPE-I 型虚拟机的初衷。本文的思路是尽量地使用客户机操作系统的现有功能。以 Linux 系统为例，现已形成了比较完善的 CPU 调频算法，并基于 Linux 系统的强生态特性，可以适配绝大部分的 CPU 调频驱动。如果要使用操作系统自己的 CPU 调频系统，最重要的是要解决特权客户机无法感知其它客户机和实际 CPU 负载的问题。

首先是关键指令探测，以 Ondemand (Linux 系统负载动态调整频率策略)调频策略为例。通过选定的 governor (调频控制策略)去调用 od_dbs_update 函数去执行定期的负载检测和频率调整。其中 od_dbs_update 调用 od_update 函数，在 od_update 中调用真正的负载计算函数 dbs_update，然后根据负载计算结果去调用调频驱动，最终改变 CPU 的硬件频率。所以我们这里需要打桩的函数是 dbs_update。使用内核提

供的 kallsyms_lookup_name 函数找到 dbs_update 的内存地址，该函数是利用内核中 kallsyms 功能。内核将系统中用到的函数符号和地址存储到 proc 文件系统中去，然后通过 kallsyms_lookup_name 函数就可以找到内核符号和地址的对应关系。然后进行负载计算重定向，在该地址上进行指令替换，将函数替换为 ARMv8 的 HVC 指令，通过该函数可以陷入 EL2 模式下的 Hypervisor 代码。由于 Hypervisor 中掌握了真正的 CPU 硬件资源，所以我们可以在这里实现实际物理 CPU 的负载计算。Hypervisor 中的负载计算模块将计算结果在虚拟化层返回(通过 eret 指令返回之前的模式)的时候传递给特权客户机。特权客户机根据计算结果调用调频驱动，实现 CPU 的动态调频。Ondemand 会根据采样频率周期性地去调节 CPU 的频率，而每次调频都会触发到 Hypervisor 中去。这样虽然是借用特权客户机的调频系统，但其实际的决策参数(负载)是由 Hypervisor 提供的，保证了 dom0 能客观地完成调频工作。最后 Hypervisor 中负载计算模块的实现。在 Hypervisor 中为每个物理 CPU 维护了一个 VCPU 的队列，VCPU 按照优先级递减的顺序依次排列在队列中。同时，每个物理 CPU 还有一个 IDLE VCPU，它是优先级最低的 VCPU 任务，在 CPU 空闲时调度，负责对 CPU 状态的一些信息统计和数据收集工作。在这些信息中包含了 IDLE VCPU 的运行时间的统计。IDLE VCPU 的运行时间就是整个物理 CPU 的空闲时间。则负载率计算如下：

$$\text{负载率} = \frac{\text{采样周期} - \text{IDEL VCPU运行时间}}{\text{采样周期}}$$

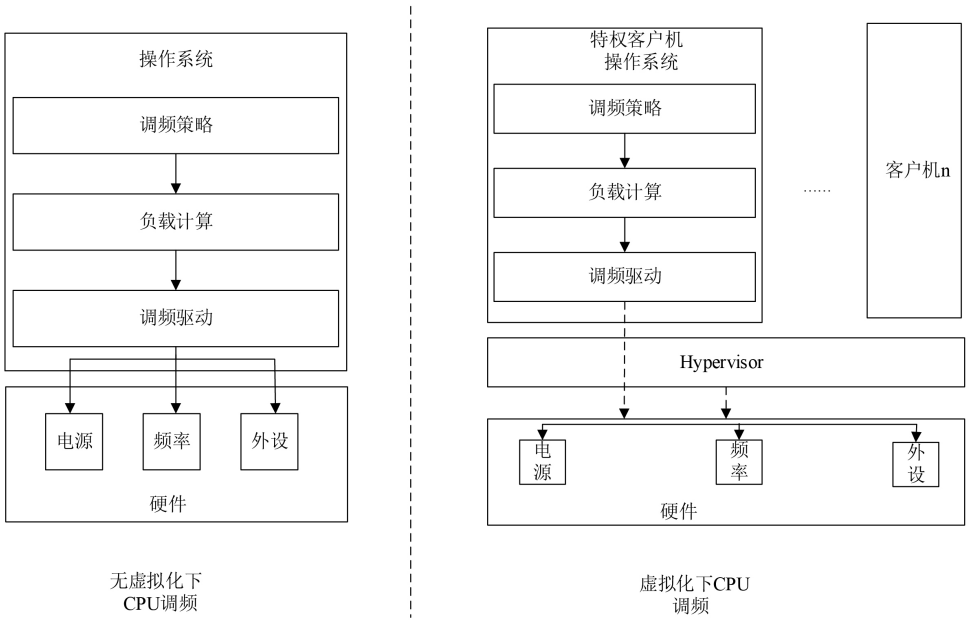

Figure 3. Diagram of CPU frequency modulation after virtualization
图 3. 虚拟化以后 CPU 调频对比图

5. 解决方案及应用案例

5.1. 采用静态隔离虚拟化的电力通信 TTU

一般来说电力网关、数据采集器(DPU)、TTU 等设备为了采集更多的设备信息就会设计多路高速串口。例如现场中遇到很多 TTU 都设计为 32 路串口或者更多。而这些串口通过总线与外部通信的波特率很高，最快的达到 2M bps。这就给 CPU 和操作系统的实时性带来很大的挑战，极易发生丢包等问题。另外这些终端和网关设备除了高速通信以外，还要负责网络通信、人机交互、本地遥测、日志存储等其它

复杂功能。传统方案一般使用实时 Linux 系统作为这些终端设备的操作系统。但是 Linux 本身的实时性较差，即使是经过实时补丁修正以后也不具备硬实时的能力。尤其是当系统负荷较重时，对实时性的影响更加严重。

对于这样实时性要求严苛且不涉及硬件资源的灵活调度的场景下，特别适合静态隔离虚拟化。它可以解决电力通信场景下，多路高速串口通信实时性不足的问题。通过改进设备的操作系统，使其具备 Linux 系统丰富生态和强大功能，同时具备更强的实时性，保证通信的安全可靠。

通过静态隔离虚拟化将 Linux 系统和 RTOS 系统混合部署到 SOC 上并将系统分为功能域和实时域。功能域运行 Linux 系统，负责设备常规的非实时业务，实时域运行 RTOS 系统进行高速串口的采集。RTOS 系统通过中断 + 轮询的方式处理大量的串口数据，并将串口数据按照串口 ID 写到共享内存的对应通道。当共享内存通道中的数据达到一定阈值以后触发 sgi 中断，通知 Linux 侧读取串口数据。这样做最大的好处是阻止了 Linux 系统因访问串口设备而造成的频繁的系统陷入以及频繁的中断造成的异常切换，这将大大减少了操作系统的损耗，增强系统实时性和减少 CPU 的占有率。Linux 侧通过编写一个虚拟串口驱动，将共享内存通道作为虚拟串口的设备层，通过平台总线设备驱动封装以后变成一个 Linux 系统的正常串口。这样原来的 Linux 侧业务就能无缝衔接过来。

5.2. 采用动态调度虚拟化的基站冗余备份

基站冗余备份是移动通信系统中确保服务可靠性和连续性的关键技术。传统方案是双机热备， 即采用两台或多台硬件设备，处于主备模式。主基站负责正常业务处理，备用基站在主基站出现故障时接管其任务。使用嵌入式虚拟化技术，将基站单元分离为不同的虚拟机。在一个虚拟机出现问题时，可以快速切换到备份的虚拟机。在正常情况下，备用虚拟机没有太多的 CPU 负载。这时可以通过调度虚拟化为其分配较少的 CPU 和内存资源。只有发生主备切换时才会被调度更多的硬件资源。

主备切换允许几百毫秒甚至秒级的延时，使用动态调度虚拟化在进行实时增强以后，完全可以达到这个指标。同时允许硬件资源动态调整，备份虚拟机只需要占用很少的资源就可以完成信息同步工作，提高了硬件利用率。

6. 总结和展望

1) 未来需要一套可在静态隔离与动态调度之间平滑切换的策略框架，避免粗粒度模式切换带来的抖动与延时。通过 tickless (NO_HZ) [5]和实时守护功能，在静态隔离时关闭调度时钟，减少周期性唤醒与 OS 抖动；在动态调度时再按需恢复时钟与抢占；实时性守护则以周期性验证 vCPU 周期/配额达成率，配合迟滞(hysteresis)逻辑抑制“频繁摆动”。

2) GICv4/v4.1 直注：利用 vPE 映射与 doorbell 机制，将外设 LPI 直接注入目标 vCPU，绕开 Hypervisor 的中断射线与 LR 编程开销，显著降低注入延迟与 VMEXIT 次数[6]。

参考文献

[1] 吕孟军. Xvisor 虚拟机管理器[D]: [硕士学位论文]. 合肥: 中国科学技术大学, 2023.

[2] 黄宇飞. ARM 平台下的虚拟化实现及应用[D]: [硕士学位论文]. 武汉: 华中科技大学, 2019.

[3] Arm System Memory Management Unit Architecture Specification SMMU Architecture Version 3. https://developer.arm.com/documentation/ihi0070/latest/

[4] Approach for CPUFreq in Xen on ARM. https://static.sched.com/hosted_files/xensummit18/ec/xen_summit_cpufreq_on_arm.pdf?_gl=1*1ae-owoq*_gcl_au*NTM4ODcyNjAwLjE3NjA5MzEyNTA.*FPAU*NTM4ODcyNjAwLjE3NjA5MzEyNTA

[5] Linux Kernel Documentation—NO_HZ: Reducing Scheduling-Clock Ticks.

https://docs.kernel.org/timers/no_hz.html

[6] GICv4.—Direct Injection of Virtual Interrupts.
https://developer.arm.com/documentation/107627/0102/GICv4-1---Direct-injection-of-virtual-interrupts

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(3), 169-175
Published Online June 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.23014

Hans汉斯

人工智能发展下的嵌入式系统教学与人才培养探索

严海蓉[1]，朱绍涛[1]，刘　钊[2]

[1]北京工业大学计算机学院，北京
[2]沧州师范学院计算机科学与工程学院，河北 沧州

收稿日期：2025年8月25日；录用日期：2025年10月20日；发布日期：2025年10月29日

摘　要

人工智能飞速发展必然带动嵌入式系统的发展。人工智能与嵌入式系统的融合的人才有哪些特点，该如何培养是当前高校所面临的问题。本文分析了人工智能与嵌入式系统的融合人才的特点：系统优化思想、具备神经网络新知识、交叉学科创新、跨学科自学等。再针对这些特点，提出了建议的新课程体系和一些可行的培养方法。

关键词

人工智能，嵌入式系统，人才培养，教学改革

Exploration of Teaching and Talent Cultivation in Embedded Systems under the Development of Artificial Intelligence

Hairong Yan[1], Shaotao Zhu[1], Zhao Liu[2]

[1]School of Computer Science, Beijing University of Technology, Beijing
[2]College of Computer Science and Engineering, Cangzhou Normal University, Cangzhou Hebei

Received: August 25, 2025; accepted: October 20, 2025; published: October 29, 2025

Abstract

The rapid advancement of artificial intelligence is bound to drive the development of embedded systems. What are the characteristics of talents integrating artificial intelligence and embedded systems, and how to cultivate such talents has become a challenge for higher education institutions.

文章引用：严海蓉, 朱绍涛, 刘钊. 人工智能发展下的嵌入式系统教学与人才培养探索[J]. 嵌入式技术与智能系统, 2025, 2(3): 169-175. DOI: 10.12677/etis.2025.23014

This paper analyzes the characteristics of talents needed for the integration of artificial intelligence and embedded systems, including systems optimization thinking, knowledge of neural networks, interdisciplinary innovation, and cross-disciplinary self-learning. Based on these characteristics, a proposed new curriculum system and some feasible cultivation methods are suggested.

Keywords

Artificial Intelligence (AI), Embedded Systems, Talent Cultivation, Teaching Reform

1. 前言

随着近年来人工智能的快速发展，嵌入式系统的教学和人才培养也提出了新的挑战。人工智能(Artificial Intelligence)，英文缩写为 AI，可以理解成仿人智能。人工智能是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新技术科学。人工智能是计算机科学的一个分支，它企图了解智能的实质，并生产出一种新的能以人类智能相似的方式做出反应的智能机器，该领域的研究包括机器人、语言识别、图像识别、自然语言处理和专家系统等[1]。

人工智能技术的迅猛发展产生了巨大人才缺口，使得培养该领域人才的任务变得紧急。很多高校从2024 年以来都逐渐开出了人工智能通识课。多地教育厅已发文，要求 2025 年秋季起，把“人工智能通识课”列为全体本科新生必修课。甚至中小学也在抓紧部署人工智能课程，教育部基础教育教学指导委员会发布《中小学人工智能通识教育指南(2025 年版)》和《中小学生成式人工智能使用指南(2025 年版)》，推进人工智能全学段教育。

嵌入式系统(Embedded System)本身就是以应用为中心，刚好为 AI 提供了“物理载体”，而 AI 则赋予嵌入式系统“智能灵魂”。二者的结合正在重塑物联网、机器人、汽车、医疗等领域的底层技术逻辑。嵌入式系统所面临的针对人工智能的改革其实是相关具身智能、机器人、智能化的具体的应用需求。

由此带来未来发展的“人工智能 + 嵌入式”人才巨大缺口，嵌入式系统教学首当其冲地必须担负起这样人才的培养责任。

在新工科建设背景下，传统嵌入式课程存在内容单一、实践不足、与 AI 脱节等问题[1] [2]，亟需构建兼顾 AI 算法与嵌入式软硬件的交叉培养体系。但嵌入式系统发展也没几年，人工智能技术迅猛发展也是近几年的事，世界高校都面临同样的人工智能技术爆发浪潮，没有较多海内外经验可以借鉴。本文仅鉴于国内外的一些高校做法[1]-[4]和作者个人的研究和项目经验，对其进行总结和归纳提炼，归纳出人工智能对嵌入式人才的需求点，从而探讨出一套人工智能 + 嵌入式系统的教学体系。

2. 人工智能发展拉动嵌入式人才培养

近年来各种新学科的爆发和繁荣，究其起源，都来自嵌入式系统和物联网学科的快速发展。嵌入式系统是支撑物联网的底层，承担着对物理世界信息的采集、传送和改变的作用，对应于传感器、无线网络和执行机构技术的发展而发展。这些底层系统的建设，使得物理世界到虚拟世界映射时每天的数据量惊人，从而促进了“云－边－端”系统框架、云计算、大数据、人工智能和机器学习等学科的快速发展。这些学科的发展最终必将使得“元宇宙”的实现越来越近。

如图 1 所示一样，嵌入式系统对其他学科而言是树干部分，每个学科都是树的分叉部分，每个叶子或者果实都是该学科的某个研究领域。果子或者叶子的快速增长，必然对树干提出新的养分输送要求，就如对人才输送的要求一样。嵌入式学科的人才对于支撑各学科发展都至关重要。

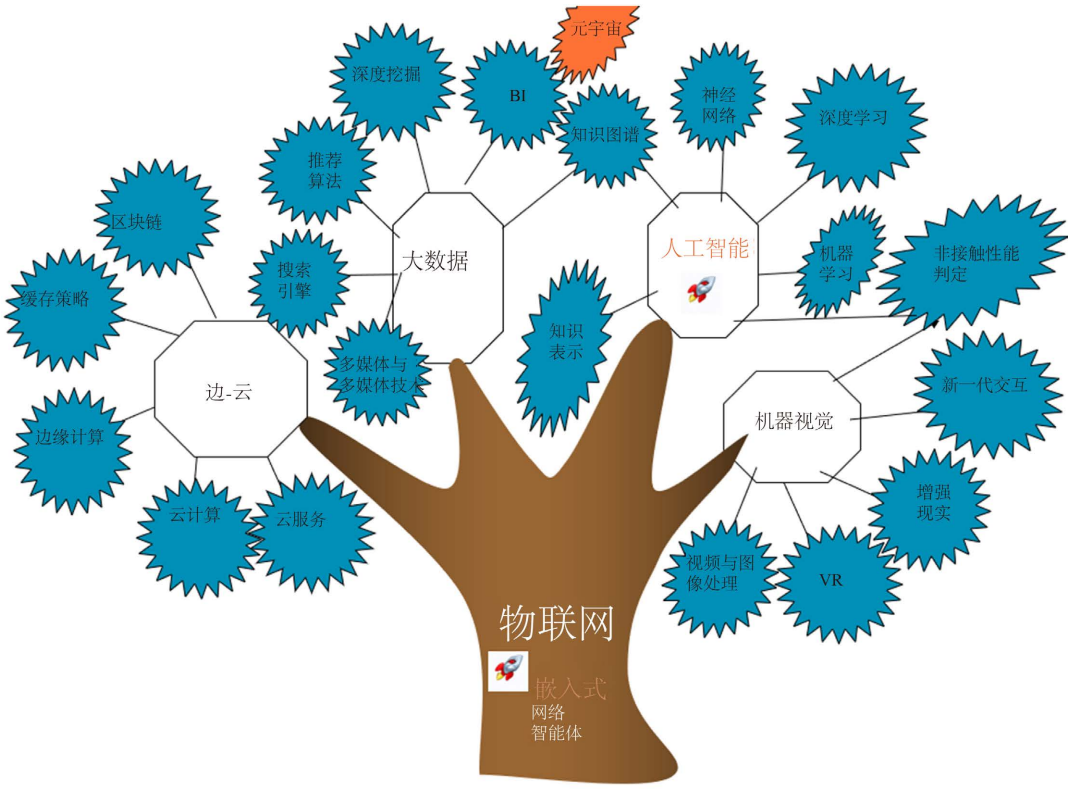

Figure 1. Interrelationship among different disciplines
图 1. 学科之间的相互关系图

2.1. 人工智能对嵌入式系统提出的新人才需求：系统优化人才

人工智能是来自对人的智能的模仿。内观我们人的智能，基本可以分为“感知－预测－判断－行动”4 个大部分。

比如对于手快被火柴烧到这个示例。眼睛会感知到“火焰”；然后会根据个人的经验内在模型产生预测，“火焰是否会烧到手，还有多久烧到？”；然后判断，是要马上吹灭火，还是扔掉火柴，还是继续观察时机？；最后执行行动，吹灭火。

该示例对应到人工智能来说，就需要用机器视觉学科技术来“看到”，然后根据某个模型(数学模型、神经网络模型等)产生预测，然后再用决策学科的某项技术进行判断，然后再利用比如云、边、端框架来传输执行某个指令。

在上面的示例中，归结到硬件上感知需要摄像头、DSP 芯片；预测模型和决策模型需要计算芯片、计算设备；执行需要执行器、各类机器人等。这些都是嵌入式系统从软件到硬件的配合。而这些其嵌入式需要系统的优化来完成整体目标，为此需要系统优化人才。

2.2. 从机器视觉发展看，需求神经网络等新知识人才

人的感知有 60%以上来自视觉，因此视觉对于人工智能来说也是非常重要的部分。下来沿着机器视

觉的发展来说明人工智能对机器视觉的影响，从而也描述如何影响到嵌入式系统。

Scene analysis (in early years)、Image understanding、Robot vision、Machine vision、Computer vision、Deep Learning for Computer Vision，这些英文名词也说明了机器视觉的发展历程。从最早在工厂场景中发展出来让机械臂和机器人对周围环境的理解，到图像理解，到机器人视觉，到机器视觉到计算机视觉到目前的深度学习机器视觉的发展过程。

对目前国外的图书调研分析如表 1。

Table 1. Analysis of foundational texts in machine vision
表 1. 基本机器视觉领域的图书分析

图书名/课程名	作者	出版年份	主要目的	图书内容
Robot vision	伯特霍尔德·霍恩	1986 年	生成一个关于被成像物体(或场景)的符号描述	图像处理、模式分类、场景分析
Image process, Analysis and Machine vision	米兰·桑卡(Milan Sonka)等	1993 年	电子化感知和理解图像复制人类视觉的效果	数字图像处理、计算机视觉、图像分析与理解
Deep Learning for computer vision (2025 斯坦福课程)	Ian Goodfellow and Yoshua Bengio and Aaron Courville	2025 年	深度理解图像并能人工合成生成图像	图像基础理论、深度学习、目标检测与识别、图像生成与风格迁移、典型应用

通过表 1 可以看到，针对机器视觉领域的知识点在不断扩大，关于神经网络、深度学习、图像生成等的人工智能新研究都在不断扩展进入其中，为此，和机器视觉主要相关的摄像机的硬件技术等虽然也在变化，但近几年来没有与软件相关的技术发展更快，融入新知识更多。

从以上分析可以看出，嵌入式被人工智能推动后，更需要了解神经网络等模型新知识的人才。

2.3. 从项目经历看，需要学科交叉人才(相关学科交叉) + 自学型人才(跨学科应用)

从个人的科研项目经历，做过“牛脸识别系统”、“混凝土性能识别系统”、“工厂 MES 工作单元系统”、“蕾丝纺织缺陷检测系统”和“心血管 IVUS 检测系统”等项目。

这些项目的共性是都有各自的学科背景，比如畜牧业、混凝土行业、工业、医疗等。在这些项目里首先需要跨学科地学习一些基础知识，这就需要有能够快速学习的自学型人才。跨学科是需要了解和与其他学科人员能够顺利沟通。

其次，这些项目都有“云、边、端”的划分。云边端发展需要学科交叉人才，比如在“云”的开发过程需要有服务器与嵌入式交叉型人才，能够将服务器、云计算知识与嵌入式相连，设计出好用的云端系统；“端”则需要自动控制与嵌入式交叉人才，既要了解机械部分又要对嵌入式非常了解；“边”则需要计算机或者单板机与嵌入式交叉。交叉的目的是要创新融合出新的解决方案。

比如随着人工智能的发展，交叉嵌入式后，“边”这里就既有人工智能驱动的边缘计算，也要有边缘计算服务的人工智能。两者侧重点不同，但都会产生新的技术，比如人工智能驱动的边缘计算就催生用户迁移技术发展，边缘计算服务的人工智能就催生对模型分发的研究。

而这些交叉人才的需求，侧重的是人才的知识学习和创新能力。

2.4. 所需要人才的知识面更广

同时，根据这些项目，由于人工智能对软件发展更有冲击力。所需要的软件人才培养的目标可以定位为面向智能终端、自动驾驶、工业控制等场景，培养既掌握人工智能算法，又具备嵌入式系统硬软协

同设计能力的复合型工程师。毕业生应能完成“算法建模→模型压缩→芯片/板级部署→实时优化”全栈开发。

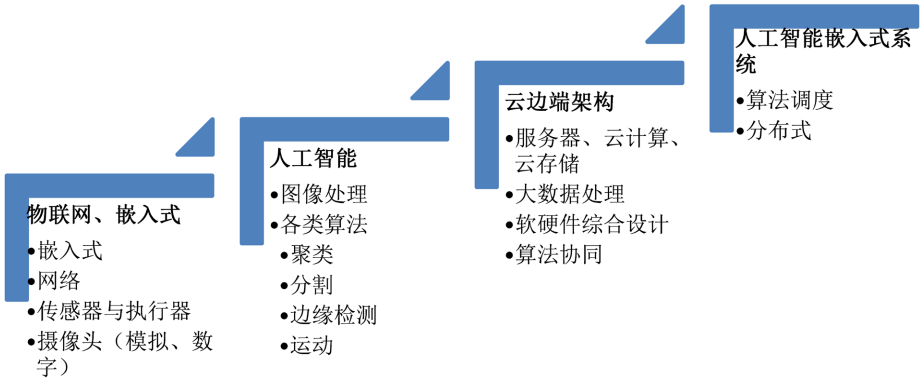

Figure 2. Knowledge requirements for software-focused talent derived from project analysis
图 2. 针对所做过项目归纳出所需要的偏软件人才培养所需要的知识面

如图 2 所示，这些知识面，明显跨度很大，交叉了很多学科。已不是原来传统的计算机专业、嵌入式专业、电子专业所能覆盖了。

3. 面向人工智能发展对人才的要求和一些应对方法

基于以上的需求，根据多年的教学经验，提出一些应对方法。

1) 一套人工智能与嵌入式融合的课程体系

虽然未来是需要“AI 原生嵌入式”人才的，但人才培养系统的变动不可能全盘从头再来。只能是基于目前的课程体系，增加有一些人工智能特点的课程，来与原来嵌入式融合。

学校可以根据现有的课程体系，适当增加三个部分：电类理解基础课、软件理解基础课与人工智能特色专业课程的相关课程，以适应人工智能和嵌入式融合发展。如图 3 所示。

“人工智能导论”课程目前已成为各专业的通识课程，在一年级开设。建议同时开设“Python 程序设计”，以便于学生们进行人工智能的实践。

在开设“嵌入式微处理器”“嵌入式系统”和“嵌入式操作系统”同时，开设“最优化方法”，培养学生最优化的思想。

在开设“模式识别与机器学习”课程后可以开设“计算机视觉”，然后以“边缘计算与嵌入式 AI”为后续融合课程。

最后在综合设计里增加“FPGA 的软硬件综合设计”让学生了解软硬一体、互相替换的系统性思想。

“大语言模型原理与应用”与“工业机器视觉”是扣紧时下新技术发展的扩展知识课程。以保持整个课程体系与现时科学发展、社会热点一致。

软件理解与电类理解课程是选取的成系统的核心骨干课程构成。可以根据本专业的性质选择偏软或者偏电类，对自己的课程体系进行补充完善。

2) 新型的课程教材和 AI 辅助教学方便学生自学

目前看到比较好的新型教材有清华大学出版社的新形态教材。该类教材在原来纸质书的章节上印上二维码，方便学生学习时候通过微信“扫一扫”，直接链接到视频课程。

同时，很多高校也在开展 AI 的辅助教学研究，产生了一大批 AI 或者仿真课程。学生可以通过相关的课程网站来学习。

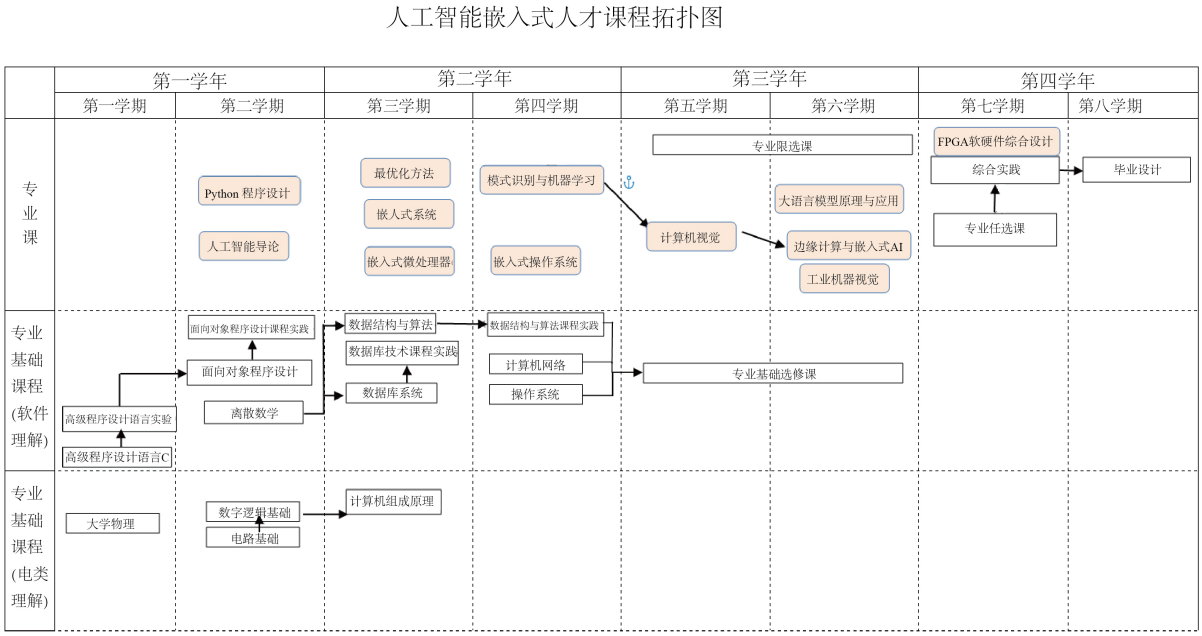

Figure 3. A proposed curriculum structure for AI-embedded systems talent
图 3. 一个可能的人工智能嵌入式人才课程拓扑图示例

这些方式，可以助力学生自学知识点，是很好的课堂辅助。

3) 多课程知识连接的知识图谱课培养学生学科交叉意识

例如将“操作系统”与“嵌入式操作系统”课程通过对知识点梳理，构建了知识图谱课程，通过知识图谱，使得学生可以在学习某个知识点的同时，关联到其他相关知识，选取自己喜欢的学习方式漫游到其他知识点。该方式下 每个同学的学习路径可以不同，从而补充了课堂知识传授的单一路线问题。让同学们自己选择缺失知识进行学习，增加学习效率。

4) 竞赛学分制带动培养自学学习习惯和创新意识

竞赛是一种很好的驱动自主学习的方式。很多同学通过竞赛会带动自己学习课堂以外的知识，同时也通过竞赛将课堂的知识进行串联，从而构建成为自己的个人的知识体系。

而竞赛获奖一定需要有创新性，也会激发同学们对创新的思考，这也是一种创新意识的培养。将竞赛放进学分系统，学生参赛将获得毕业必须的创新学分，从而也保证了全体学生的积极参与。

4. 结论

我国该方面人才培养的总的思路是从“嵌入式 + AI”到“AI 原生嵌入式”人才培养，以匹配我国该方面技术国产替代、自主研发的人才需求。人工智能融合嵌入式带来人才在具备系统优化思想、能够交叉学科创新、拥有神经网络新知识、具有强自主学习能力等方面的人才培养需求。为此，课程体系的改变是必须的，但仅仅依靠课程是不足的。新的课程手段比如新形态教材、知识图谱课程、竞赛学分制等方式是以上人才培养的一个可行之路。

参考文献

[1] 管芳, 刘涛, 赵中华, 等. 新工科背景下的嵌入式系统课程教学改革研究[J]. 大学教育, 2025(6): 55-60.

[2] Piedad, E.J. (2024) Strengthening Artificial Intelligence-on-Edge Education in the Philippines: A Teacher-Centric Curriculum Development Strategies. 2024 *IEEE* 13*th International Conference on Engineering Education* (*ICEED*), Kanazawa,

19-20 November 2024, 1-6. https://doi.org/10.1109/iceed62316.2024.10923809

[3] 毕盛, 董敏, 冼进, 汪秀敏. 结合人工智能技术的嵌入式系统教学改革[J]. 当代教育实践与教学研究, 2024(10): 73-76.

[4] Mo, T., Li, W. and Zhang, L. (2025) Teaching Practice and Innovation in the Professional Master's Degree Program in Electronic Information at Peking University in the Era of Artificial Intelligence. 2025 *IEEE International Conference on Software Services Engineering* (*SSE*), Helsinki, 7-12 July 2025, 226-232. https://doi.org/10.1109/sse67621.2025.00037

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(3), 176-195
Published Online June 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.23015

Hans汉斯

RusT-Thread：基于Rust面向资源受限嵌入式设备的操作系统的实践

罗浩民，陈琳波，刘　时，李　丁，赵于洋

中国科学技术大学计算机科学与技术学院，安徽　合肥

收稿日期：2025年9月22日；录用日期：2025年10月21日；发布日期：2025年10月30日

摘　要

随着物联网和嵌入式系统的发展，实时操作系统(RTOS)的安全性和性能需求日益提高。传统基于C语言的RTOS在内存安全和并发控制方面存在局限，容易导致缓冲区溢出、数据竞争等问题。本项目以RT-Thread为基础，使用Rust语言重构其内核，形成了全新的RusT-Thread系统。系统采用模块化架构，涵盖内核服务、进程调度、内存管理、线程通信与时钟控制等核心功能，并充分利用Rust的所有权模型与类型系统，实现内存安全与并发安全保障。项目创新性地引入改进的多级反馈队列调度算法、中断安全数据容器(RTIntrFreeCell)、内联汇编与动态－静态数据分离等技术，在保证功能兼容性的同时优化了代码简洁性与可维护性。通过单元测试、集成测试和性能基准测试，RusT-Thread在中断延时、上下文切换和线程创建时间等关键指标上表现出与RT-Thread相当甚至更优的实时性能。该工作不仅展示了Rust在系统软件开发中的可行性与优势，也为未来安全可靠的嵌入式RTOS设计提供了参考。

关键词

Rust，RTOS，RT-Thread，内存安全，并发安全，多级反馈队列调度，嵌入式系统

RusT-Thread: A Rust-Based Operating System for Resource-Constrained Embedded Devices

Haomin Luo, Linbo Chen, Shi Liu, Ding Li, Yuyang Zhao

School of Computer Science and Technology, University of Science and Technology of China, Hefei Anhui

Received: September 22, 2025; accepted: October 21, 2025; published: October 30, 2025

文章引用: 罗浩民, 陈琳波, 刘时, 李丁, 赵于洋. RusT-Thread: 基于 Rust 面向资源受限嵌入式设备的操作系统的实践[J]. 嵌入式技术与智能系统, 2025, 2(3): 176-195. DOI: 10.12677/etis.2025.23015

Abstract

With the rapid development of IoT and embedded systems, the requirements for the security and performance of real-time operating systems (RTOS) are increasing. Traditional C-based RTOS implementations suffer from limitations in memory safety and concurrency control, which can lead to buffer overflows, data races, and system instability. This project reconstructs the RT-Thread operating system entirely in Rust, resulting in a new RusT-Thread system. The system adopts a modular architecture covering kernel services, process scheduling, memory management, inter-thread communication, and clock control, while leveraging Rust's ownership model and type system to ensure memory and concurrency safety. Key innovations include an improved multi-level feedback queue scheduling algorithm, an interrupt-safe data container (RTIntrFreeCell), inline assembly integration, and a dynamic–static data separation design, which optimizes code simplicity and maintainability while ensuring functional compatibility. Comprehensive validation through unit tests, integration tests, and performance benchmarks demonstrates that RusT-Thread achieves real-time performance comparable to or even better than RT-Thread in terms of interrupt latency, context switching, and thread creation time. This work highlights the feasibility and advantages of Rust in system software development and provides valuable insights for the design of secure and reliable embedded RTOS in the future.

Keywords

Rust, RTOS, RT-Thread, Memory Safety, Concurrency Safety, Multi-Level Feedback Queue Scheduling, Embedded Systems

1. 简介

RusT-Thread 是一款基于 RT-Thread 理念，采用 Rust 语言打造的轻量级实时操作系统内核，已在 ARM Cortex-M4 上成功实现，并有望拓展至更多芯片平台。

在线程调度上，它支持多线程创建、调度及优先级管理，提供多种调度算法，如优先级、优先级 + RR 等。同时，在内存分配上支持伙伴系统、小内存分配器等多种内存分配方式，满足动态内存分配需求。此外，还实现了高精度定时器，支持单次和周期定时功能，并具备完整的异常和中断处理机制。

RusT-Thread 的 Rust 实现保障内存安全，模块化设计便于扩展和移植，支持资源受限的嵌入式系统，为开发者提供了安全、高效、精简的 Rust 原生实时操作系统选择。

本文将从项目特色、架构设计、核心实现机制、性能验证等多个维度，详细阐述 RusT-Thread 的设计理念与实现方式，展示其如何将 Rust 的现代化语言优势与实时操作系统的经典思想相结合，为嵌入式开发者提供一个更安全、更高效的开发新选择。

2. 项目特色

本项目是一个完全由 Rust 构建的操作系统内核。不同于其他使用 Rust 重构操作系统的工作，我们放弃了 C 和 Rust 混合编译改写操作系统内核的开发路径，而是选择从底层开始就用 Rust 开发，这种彻底的重构能让我们更好地利用 Rust 拥有的现代化语言特性，组织起结构更加清晰，功能实现更加简练的代

码。同时，我们还避免了混合编译过程中的各种操作性问题和潜在的安全性风险。

本项目的架构参考 RT-Thread nano 内核实现，在实现功能模块时我们保留了大部分 RT-Thread 内核的接口[1] [2]，确保熟悉 RT-Thread 内核的开发者能低成本快速上手我们的内核。RT-Thread 的成功很难离开其众多贡献者带来的丰富软件包移植，我们希望我们的内核也能为 RT-Thread 社区中的 Rust 开发者提供一个底层平台，可以原生地用 Rust 实现各种组件，丰富这个操作系统的功能。

3. 架构概述

当我们着手设计 RusT-Thread 架构时，我们的目标并不仅仅是复刻 RT-Thread，而是要用 Rust 语言的思想去重塑它，我们设计的总体架构如图 1 所示。为此，我们将以下几个关键原则融入了项目里：

- **安全性**：借助 Rust 的所有权系统与借用检查机制，从编译阶段就彻底消除内存安全隐患。同时，通过其严格的并发模型有效防止数据竞争，能够有效提升系统的并发安全性。
- **可扩展性**：我们将整个系统设计为高度模块化。这种架构使得无论是添加新的设备驱动、文件系统，还是集成复杂的网络协议栈，都变得直观而高效，为未来的功能拓展留出了充足的空间。
- **性能**：充分利用 Rust 的零成本抽象特性，在确保系统安全性的基础上，对调度算法和内存管理机制进行深度优化，从而全面提升系统的整体性能表现。
- **易移植性**：我们采用分层设计策略并构建清晰的硬件抽象层，简化了系统对不同芯片架构的适配过程。目前，该系统已成功支持 ARM Cortex-M4 芯片架构，未来也许会逐步扩展对更多类型芯片的支持范围。

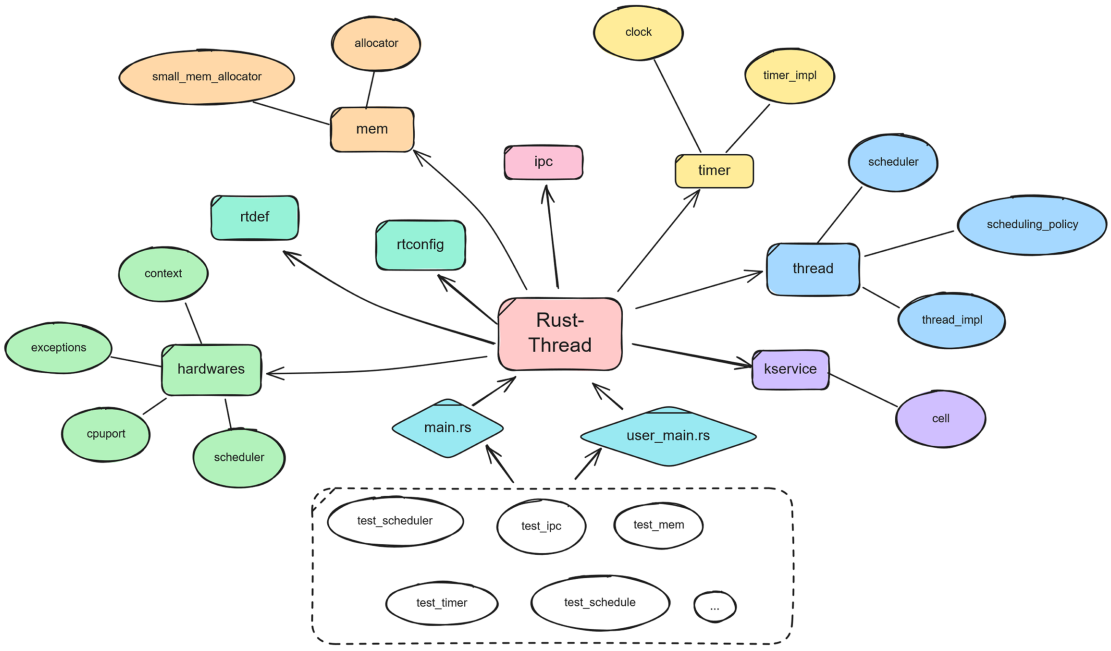

Figure 1. RusT-Thread overall architecture
图 1. RusT-Thread 总体架构图

4. RusT-Thread 模块介绍

4.1. 内核服务层

内核服务层作为 RusT-Thread 操作系统的核心基础，是连接硬件底层与上层应用的关键桥梁，其核

心功能及在 Rust 中的实现方案如下：

- **错误处理机制**：采用 Rust 的 Option 和 Result 枚举类型替代传统错误码，通过模式匹配清晰表达函数执行结果。在编译期对错误处理逻辑进行严格检查，避免因错误处理不当引发的问题，确保系统稳定运行。
- **内存操作函数**：利用 Rust 标准库中的 Core::alloc 模块，结合自定义的内存分配策略和数据结构，实现高效灵活的内存分配与回收机制，优化内存使用效率，确保操作的正确性和安全性。
- **字符串操作函数**：借助 Rust 内置的 String 和 str 类型及其操作方法，实现功能强大且安全高效的字符串处理功能，有效防止缓冲区溢出等问题，满足系统对文本处理的需求。
- **格式化输出功能**：结合 cortex_m_semihosting 工具库和 Rust 的格式化字符串宏(format!、println!等)，实现数据格式化输出功能。在 QEMU 模拟器环境下，通过半宿主功能将输出数据发送至宿主机控制台，支持多种数据类型和输出格式，满足调试和信息展示的需求。
- **调试和断言功能**：借助 Rust 的 Debug 和 Display 特性以及 assert!、debug_assert!等宏，实现完善的调试和断言功能。为自定义数据类型实现相关特质，以便在调试输出时展示详细信息；通过断言条件检查及时暴露问题，提高开发效率。
- **链表实现**：利用 Rust 的 Vec 等标准容器类型，结合手动指针操作和数据结构管理逻辑，构建高效的链表结构。通过存储节点指针模拟链接关系，实现节点的动态操作，同时借助所有权系统和借用检查机制确保操作的安全性和正确性。

我们单独添加了 Cell 模块，以更好地适配 Rust 的语言特性。Cell 模块实现了一个中断安全的共享数据容器 RTIntrFreeCell<T>，这是 RT-Thread 实时操作系统中的核心内核服务组件之一。其核心目的就是在多线程和中断环境下提供安全的共享数据访问机制，通过自动禁用和启用中断来防止数据竞争，具体体现在以下三个方面：

1. 中断安全性

- 在访问共享数据时自动调用 rt_hw_interrupt_disable ()禁用中断
- 访问结束后自动调用 rt_hw_interrupt_enable ()恢复中断
- 确保临界区代码能够原子性地执行，避免数据不一致的问题

2. RAII 资源管理

- 使用 RTIntrRefMut 作为智能指针，利用 Rust 的 Drop 特质实现自动资源释放
- 当 RTIntrRefMut 超出作用域时，自动恢复中断级别，降低了手动管理资源的风险

3. 灵活的访问方式

- 提供 exclusive_access ()方法获取独占访问权限
- 提供 exclusive_session ()方法在闭包中执行临界区代码
- 提供 as_ptr ()和 as_mut_ptr ()方法获取原始指针，方便底层操作
- 提供 field_ptr ()和 field_mut_ptr ()方法获取结构体字段的原始指针，增强灵活性

在 RusT-Thread 内核中，我们使用这个核心模块来保护那些需要同时被中断服务程序和普通线程访问的关键数据，比如线程控制块(TCB)、调度器状态、定时器列表以及内存管理的内部数据。RTIntrFreeCell<T>的作用就是提供一个“中断锁”，确保在任何时候数据访问都是安全的，这对于保证高并发下系统的数据一致性和整体稳定性至关重要。

4.2. 硬件抽象层

硬件抽象层(hardware)是 RusT-Thread 操作系统在 Cortex-M4 架构下的底层硬件支持模块，主要负责

CPU 端口、上下文切换、异常处理和中断管理等功能，为上层系统提供与硬件紧密相关的基础服务。

硬件抽象层是操作系统的核心组成部分，通过这些模块，RusT-Thread 能够实现高效、可靠的硬件资源管理和任务调度，为上层应用提供稳定的运行环境。

模块内容如下：

```
hardware
├── context.rs    # 线程上下文切换，使用内联汇编实现线程上下文保存与恢复
├── cpuport.rs    # CPU 接口-Cortex-M4，定义了异常栈帧、栈帧、CPU 关机、CPU 重启
├── exception.rs  # 异常处理相关函数
├── irq.rs        # 中断管理模块，提供了中断嵌套计数、钩子设置、使能/禁用中断等功能
└── mod.rs        # 统合上述模块，提供对外接口
```

1. 上下文切换模块(context.rs)

负责线程上下文的切换机制，主要功能包括：

- 线程上下文的保存与恢复
- PendSV 中断处理(实际执行上下文切换)
- 提供 rt_hw_context_switch、rt_hw_context_switch_interrupt 和 rt_hw_context_switch_to 等上下文切换处理函数
- 实现线程间的高效切换，是多线程调度的核心机制

2. CPU 端口模块(cpuport.rs)

提供与 CPU 硬件直接相关的底层支持[3]：

- 定义异常栈帧 ExceptionStackFrame 和栈帧 StackFrame 结构
- 实现线程栈初始化函数 rt_hw_stack_init
- 提供 CPU 关机 rt_hw_cpu_shutdown 和重启 rt_hw_cpu_reset 等功能
- 可选的 FPU 支持(浮点运算单元)

3. 中断管理模块(irq.rs)

实现中断处理相关的功能：

- 中断嵌套计数管理
- 提供中断使能/禁用函数 rt_hw_interrupt_disable/enable
- 中断进入/退出处理 rt_interrupt_enter/leave
- 支持中断钩子函数设置(通过特性开关控制)
- 获取中断嵌套层数 rt_interrupt_get_nest

4. 异常处理模块(exception.rs)

负责处理系统运行中的各种异常：

- 提供硬件错误处理(HardFault、MemManage、BusFault、UsageFault)
- 异常信息收集与输出
- 支持异常钩子机制 rt_hw_exception_install

- 详细的故障跟踪与诊断功能

4.3. 进程调度层

作为 RusT-Thread 操作系统的核心，进程调度层负责管理和调度所有线程，确保它们能够高效、公平地共享处理器资源，实现并发执行，核心职责如下：

- **任务调度**：依据预设调度策略，决定处理器执行权的分配，保障高优先级任务的及时响应，同时兼顾任务的公平执行。支持多种调度算法，具备灵活适应不同应用场景和实时性要求的能力。
- **调度算法选择**：提供优先级调度算法(如优先级 + 时间片轮转)和多级反馈队列调度算法等，用户可根据实际需求灵活选择调度算法，满足不同任务对实时性和资源分配的需求。
- **线程 API 接口**：为用户提供了一系列丰富的接口，使用户能够更加便捷地对线程状态进行调整和控制，例如创建、删除、挂起、恢复线程等操作，我们设计的线程状态转化如图 2 所示，同时提供了获取和设置线程属性的接口，方便用户根据实际需求对线程进行精细化管理。

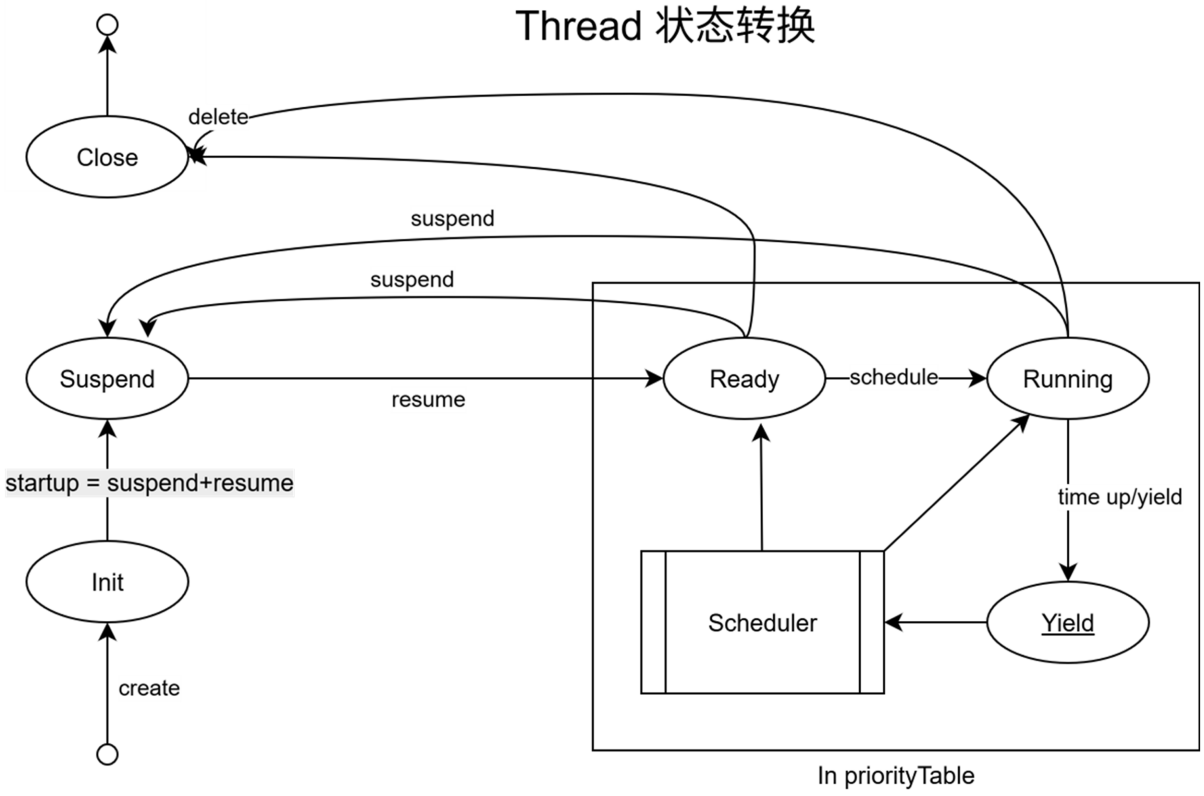

Figure 2. Process state transition
图 2. 进程状态转换图

- **调度策略抽象**：通过定义 SchedulingPolicy 特质，将多种调度算法封装和抽象。以优先级调度算法为例，实现了相应的结构体，包含线程优先级队列管理和时间片分配与轮转机制；对于多级反馈队列调度算法，设计了多级队列数据结构，实现了任务根据执行时间动态调整队列级别的逻辑。基于特质的抽象方式使调度策略切换灵活，用户在初始化时指定调度策略后，调度器即可根据具体实现执行调度操作。

```
/// 调度策略

traitpub trait SchedulingPolicy: Send + Sync {

    /// 选择下一个要运行的线程

    ///

    /// # 参数

    /// * `current_thread` - 当前运行的线程

    ///

    /// # 返回值

    /// * `Option<(Arc<RtThread>, bool)>` - (选中的线程, 是否需要将原线程重新插入就绪队列)

    fn select_next_thread(

        &self,

        current_thread: &Option<Arc<RtThread>>,

    )

    -> Option<(Arc<RtThread>, bool)>;

    /// 获取策略名称

    fn get_policy_name(&self) -> &'static str;

    }
```

- **调度算法优化：**引入位图 + FFS (Find First Set)算法提升调度效率。位图标识各优先级队列的线程就绪状态，通过 FFS 算法快速定位最高优先级队列中的首个就绪线程。FFS 算法借助预计算查找表，将复杂位运算转化为简单数组访问，实现 O (1)时间复杂度的最高优先级线程查找，显著提升调度器响应速度和实时性能，确保系统及时响应高优先级任务。

```
#[cfg(feature = "tiny_ffs")]const __LOWEST_BIT_BITMAP: [u8; 37] = [

    0, 1, 2, 27, 3, 24, 28, 32, 4, 17, 25, 31, 29, 12, 32, 14,

    5, 8, 18, 32, 26, 23, 32, 16, 30, 11, 13, 7, 32, 22, 15, 10,

    6, 21, 9, 20, 19];

#[cfg(feature = "tiny_ffs")]pub fn __rt_ffs(value: u32) -> u8 {

    if value == 0 {

        return 0;

    }

    __LOWEST_BIT_BITMAP[((value & (value - 1) ^ value) % 37) as usize]}
```

该算法利用预计算查找表和数学运算避免复杂位运算循环，实现了快速最低有效位查找。它专为嵌入式环境优化，在资源受限设备上也能快速执行，满足实时系统低延迟要求。能快速确定线程就绪队列中最高优先级线程，确保调度器及时响应高优先级任务，提升系统实时性和性能。

- **多级反馈队列调度实现：**在多级反馈队列调度算法中，引入老化机制，使长期未被调度的线程逐渐升高优先级，防止低优先级线程饥饿，增强调度算法公平性和适应性。实现过程中，借助 Rust 语言零成

本抽象特性，优化代码实现，提升系统稳定性和可维护性。

4.4. 内存管理层

在 RusT-Thread 中，我们实现了原本 RT-Thread 中的小内存分配器，同时并支持了库实现的 buddy_system allocator 和 good_memory allocator，可供用户在实际场景中自由选择，下面将对小内存分配器作具体分析：

RusT-Thread 中的小内存分配器主要体现在如下几个文件中，具体的内存数据组织方式如图 3 所示：

- small_mem_impl.rs：核心算法实现
- small_mem_allocator.rs、allocator.rs：分配器接口与适配
- object.rs、safelist.rs：辅助对象和安全链表
- oom.rs：内存溢出处理

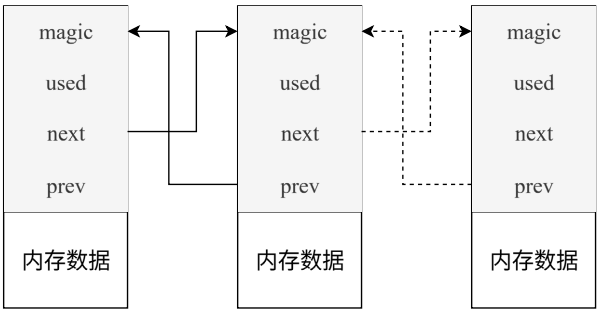

Figure 3. Memory data organization method
图 3. 内存数据组织方式

表示内存块的结构体：

```
///RTSmallMemItem 结构体表示一个小内存块的基本信息，主要用于管理内存池中的单个内存块#[repr(c)]pub struct
RTSmallMemItem {

  ///内存池指针

  pub pool_ptr: usize,

  #[cfg(target_pointer_width = "64")]// 条件编译，64 位系统生效

  pub resv: u32,//保留字段，用于对齐，64 位系统下使用

  ///下一个空闲块的指针

  pub next: usize,

  ///前一个空闲块的指针

  pub prev: usize,

  #[cfg(feature = "mem_trace")]//条件编译，内存跟踪生效

  #[cfg(target_pointer_width = "64")]//条件编译，64 位系统生效

  pub thread: [u8; 8],//线程 ID，64 位系统下使用

  #[cfg(feature = "mem_trace")]//条件编译，内存跟踪生效

  #[cfg(target_pointer_width = "32")]//条件编译，32 位系统生效

  pub thread: [u8; 4],//线程 ID，32 位系统下使用}
```

小内存分配算法原理是通过维护一块连续的内存池，将其划分为带有头部信息的内存块，并用链表管理空闲和已用块。分配时遍历空闲链表，找到足够大的块后分割并标记为已用；释放时将块标记为空闲，并尝试与相邻空闲块合并以减少碎片。整个过程包含边界检查和中断保护，确保分配、释放的安全性和原子性。

除了实现基本的小内存算法外，我们还有如下亮点：

(1) 边界检查与安全性提升

C 代码主要依赖 RT_ASSERT 等宏进行运行时断言，且大量裸指针操作，容易出现悬垂指针、越界、重复释放等问题，这些断言如果被关闭，代码安全性大幅下降。

```
debug_assert!((mem as usize) >= ((*small_mem).heap_ptr as usize));

debug_assert!((mem as usize) < ((*small_mem).heap_end as usize));

debug_assert!(mem_is_used(mem));

if m.is_null() || size == 0 {

    return ptr::null_mut();

}
```

Rust 利用类型系统和所有权机制，天然防止了大部分内存安全问题，同时 rt_smem_free、rt_smem_alloc 等函数在操作前都做了空指针和边界检查。

Rust 的 debug_assert!只在 debug 模式下生效，release 下可关闭，但类型系统和生命周期机制依然提供了额外的安全保障。

许多辅助函数(如 mem_is_used、mem_pool 等)都用 inline 和类型安全的方式实现，减少了手动错误。

(2) 中断保护

C 语言通过 rt_hw_interrupt_disable/rt_hw_interrupt_enable 手动保护关键区，防止并发破坏堆结构。

```
rt_base_t level = rt_hw_interrupt_disable();

// ...关键区.

rt_hw_interrupt_enable(level);
```

Rust 同样调用 rt_hw_interrupt_disable/rt_hw_interrupt_enable，但更易于用 RAII (资源自动释放)等机制进行封装，减少人为失误。

```
let level = rt_hw_interrupt_disable();

//...关键区 ...

rt_hw_interrupt_enable(level);
```

并且 Rust 代码结构更清晰，便于后续用 RAII 或作用域自动恢复中断，提升健壮性。

4.5. 线程通信层

进程间通信(IPC)是多任务操作系统中各个任务之间进行数据交换和协同工作的重要手段。我们的 RusT-Thread 提供了信号量机制，而消息队列、邮箱等作为拓展，我们尚未支持。

信号量工作示意图如图 4 所示，每个信号量对象都有一个信号量值和一个线程等待队列，通过信号量的值是否为零，决定线程是否可以访问临界区的资源，当信号量实例数目为零时，再申请该信号量的线程就会被挂起在该信号量的等待队列上，等待可用的资源。

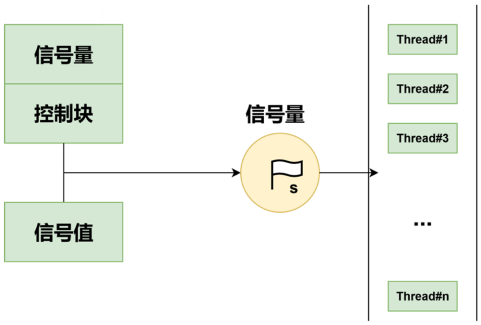

Figure 4. Semaphore working diagram
图 4. 信号量工作示意图

在 RT-Thread 中，信号量相关操作有以下函数——创建、删除、获取、释放，我们实现的思路和 C 类似，先实现 ipc 的基础操作，如_ipc_list_suspend 挂起线程，_ipc_list_resume 唤醒线程等等，ipc 相关操作函数如图 5 所示，然后，通过信号量的值，选择不同的操作，实现信号量的相关操作即可。

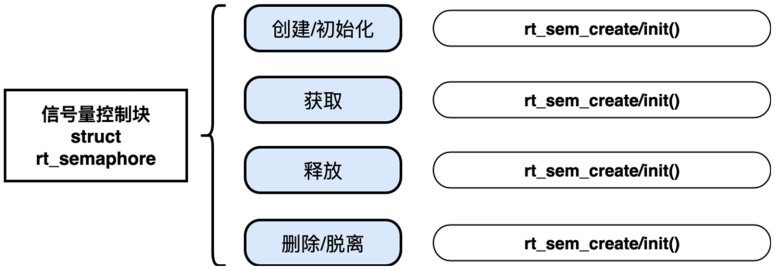

Figure 5. Semaphore related operation functions
图 5. 信号量相关操作函数

4.6. 时钟控制层

4.6.1. 时钟节拍的产生

时钟节拍由配置为中断触发模式的硬件定时器产生，当中断到来时，将调用一次 rt_tick_increase ()函数，通知操作系统已经过去一个系统时钟；不同硬件定时器中断实现都不同，Rust_Thread 的中断函数是在 QEMU 模拟器上的 stm32 系列单片机上实现的，具体地，程序中将使用#[exception]一个中断处理函数 SysTick ()，在其中调用 rt_tick_increase ()函数。

在 RusT-Thread 系统中，使用常数 RT_TICK_PER_SECOND 控制时钟周期长度。本系统中默认主频是 16 MHZ，是通过 QEMU 模拟芯片内部高速振荡器实现的，默认 RT_TICK_PER_SECOND = 1000，即一个时钟周期 16000 个硬件周期。

4.6.2. 时钟中断的管理

时钟中断管理核心函数 rt_tick_increase ()主要完成以下工作：

- 将全局变量 RT_TICK 自增，这个变量记录了系统从初始化到当前经过了多少个时钟周期，叫做系统时间。

- 检查当前线程的时间片是否到期，若到期，则触发线程调度。
- 检查是否有定时器到期，如果有，触发定时器超时函数。

时钟管理中还包括系统时间的读取和设定函数，毫秒数和时钟周期数的转换函数等功能函数。

4.6.3. 定时器的管理

RT-Thread 的定时器由定时器控制块 RtTimer 控制，定时器控制块全部在运行时动态分配内存并按照超时时间升序挂载在动态数组 TIMERS 中。定时器的管理主要包括定时器的创建、激活、修改、超时与停止。

- 创建：使用 new 方法创建，体现了面向对象的思想。
- 激活：计算超时时刻，通过二分查找插入 TIMERS 数组，保证有序性。
- 修改：使用 enum 封装所有修改操作(如周期、定时值)，符合 Rust 风格且易于扩展。修改在下次激活时生效。
- 超时：通过二分查找高效定位到期的定时器，执行回调。周期性定时器会自动重新激活，单次定时器则被移除。
- 停止：从 TIMERS 数组中移除并回收定时器。

5. RusT-Thread 性能与验证测试

5.1. 内存性能测试

原生的 RT-Thread 官方并没有给出一些具体的有关内存性能的数据，所以这里我们参照标准的 std 库来比较分析 RusT-Thread 的性能。

我们在 Linux (x86_64)平台下，使用 Rust 重新实现了 RusT-Thread 的完全相同的内存管理模块，并对其进行了适当的封装，并使其接口与原系统保持一致。这样，我们就可以在支持 std 库和 Criterion 基准测试框架的环境下，对内存分配、释放等核心操作进行高效、可重复的性能测试，并与 Rust 标准库分配器进行公平对比。

- 测试代码使用 Criterion 框架实现，测试内容包括小块分配、混合分配、碎片处理、内存利用率等多种典型场景。
- 这种方法虽然不能完全反映嵌入式平台的绝对性能，但可以有效比较不同分配算法的相对性能，为实际部署和优化提供有价值的参考。

具体测试结果如图 6 所示。

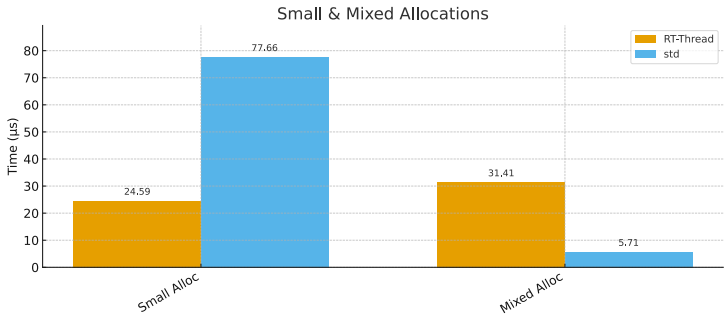

Figure 6. Memory performance test results
图 6. 内存性能测试结果

图 6 展示了 RusT-Thread 与标准分配器在进行小块内存分配和混合内存分配时的性能表现：

- Small Allocations 中，RusT-Thread 平均耗时约为 24.6 μs，明显优于 std 的 77.7 μs，表明 RusT-Thread 在频繁的小对象分配中具有更低的管理开销。
- Mixed Allocations 中，标准分配器表现优异，耗时仅约 5.7 μs，而 RusT-Thread 则为 31.4 μs，可能是由于 RusT-Thread 对变长内存块的处理不如标准库灵活高效。

RusT-Thread 在固定小块内存操作中具有优势，但在面对内存尺寸变化复杂的情况时性能下降。不过这同时也展现出不同应用场景中，RusT-Thread 中小内存分配算法的性能更加稳定。

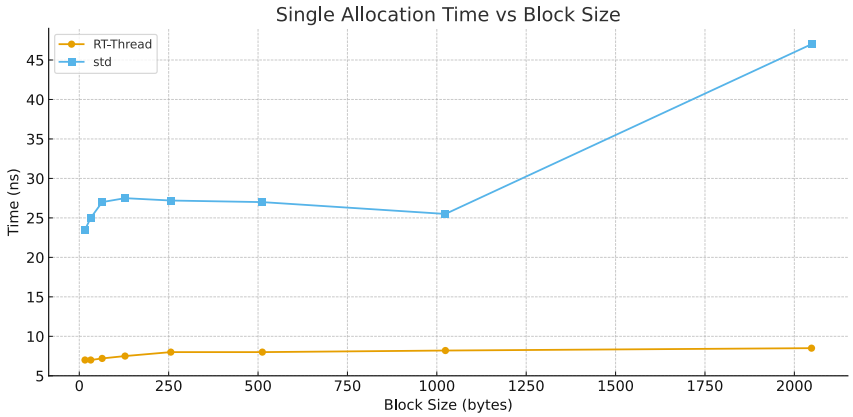

Figure 7. Trend of single allocation time changes for RusT-Thread and standard allocator
图 7. RusT-Thread 和标准分配器的单次分配耗时变化趋势

图 7 展示了在逐步增加内存块大小的条件下，RusT-Thread 和标准分配器的单次分配耗时变化趋势：

- RusT-Thread 的分配时间基本稳定在 7~8 ns 范围内，说明其设计对小中等大小块分配进行了优化处理，性能几乎不受块大小影响
- 标准分配器的耗时随着块大小增加而变化更为剧烈，例如从 16B 的约 24 ns 上升到 2048B 时超过 47 ns，说明其可能使用了更复杂的分配策略或存在内存对齐开销

可见，RusT-Thread 分配器的响应速度更稳定，适用于内存块尺寸变化不大的嵌入式任务场景。

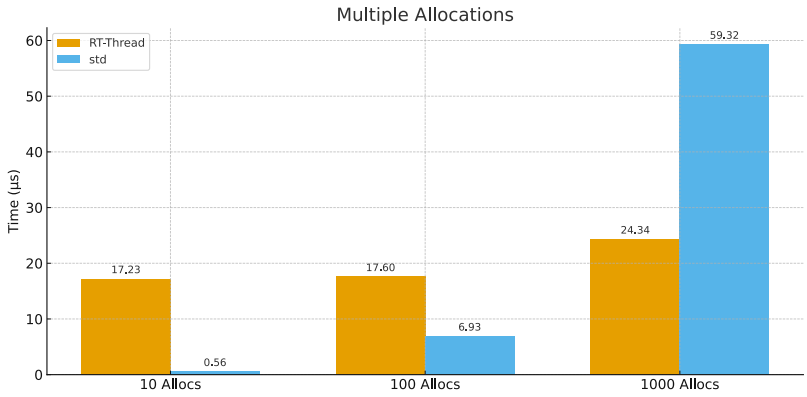

Figure 8. Average time taken by RusT-Thread allocator and standard allocator for bulk memory allocation
图 8. RusT-Thread 分配器与标准分配器在进行批量内存分配时的平均耗时

图 8 对比了在进行批量内存分配(10、100、1000 次)时，两种分配器的平均耗时：

- RusT-Thread 在 10~100 次批量分配中仅需 17 μs 左右，而标准分配器耗时逐步上升到 60 μs 以上。
- 差距在批量操作中进一步放大，表明 RusT-Thread 的批处理性能更优，内部结构对频繁申请释放有较强的适应性。

可以看出，RusT-Thread 分配器在多次重复分配的密集型任务中更具优势，特别适用于任务频繁上下文切换或数据缓冲场景。

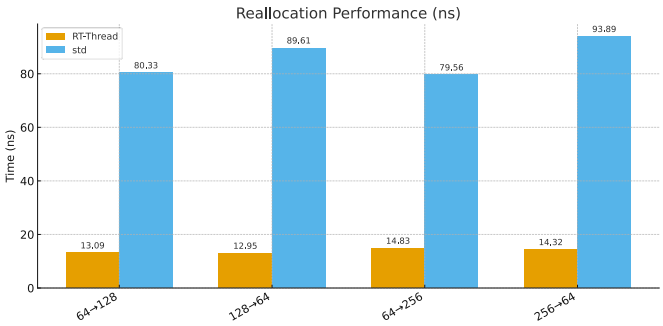

Figure 9. Time comparison under four memory reallocation paths
图 9. 四种内存重分配路径下的耗时比较

图 9 展示了常见四种内存重分配路径(如 64→128，128→64，64→256，256→64)下的耗时比较：

- 所有场景中 RusT-Thread 的耗时都远低于标准分配器，例如 128→64 仅约 12.95 ns，而标准库约 89.6 ns
- 表明 RusT-Thread 分配器可能采用就地扩展或快速搬迁机制，避免了额外的复制和元信息更新开销

所以，RusT-Thread 对 reallocation 操作进行了优化，在需要动态改变内存块大小的场景中表现更加出色，这使得对于真实环境中的复杂情况的处理会更加优秀。

5.2. 模块验证测试

测试部分代码结构如下：

```
RusT-thread/src/test
├── README.md                 # 测试/示例的说明与使用方法，概览覆盖点与调用入口
├── comprehensive_example.rs  # 全面功能演示（线程/定时器/调度策略/MFQ/中断等全量展示）
├── example.rs                # 基础功能演示入口（线程生命周期、优先级/睡眠/让出等）
├── example_mfq.rs            # 多级反馈队列（MFQ）调度策略的示例与演示。
├── integration_test.rs       # 系统级集成测试套件，覆盖线程/内存/调度/定时器/IPC，并发、压力与边界场景。
├── mod.rs                    # 测试模块聚合与导出，集中开关
├── performance_test.rs       # 性能与吞吐/时延评估，随机负载仿真、指标统计与可视化输出。
├── switch_time_test.rs       # 关注上下文切换/调度路径时间测量与分析。
├── test_all.rs               # 打印测试 Banner/Logo 的入口展示函数。
├── test_cell.rs              # 中断安全数据结构（如 RTIntrFreeCell）的并发访问测试。
├── test_excp.rs              # 异常处理相关路径的触发与校验。
├── test_interupt.rs          # 中断启停与相关功能的基础测试。
├── test_ipc.rs               # IPC 队列挂起/唤醒/全唤醒与优先级队列语义的多线程联调。
├── test_mem.rs               # 基础内存分配/释放，Vec/Box 等简单路径验证。
├── test_scheduler.rs         # 调度器插入/启动/调度行为的功能性测试。
├── test_small_mem.rs         # 小块内存分配器的专项测试（受 feature 控制）。
├── test_thread.rs            # 线程基本能力测试（创建/启动/挂起/恢复）与裸上下文切换验证。
└── test_timer.rs             # 定时器单次/周期回调与停止控制的功能测试。
```

5.3. 单元测试

我们为各个模块编写了详细的单元测试用例，对模块的功能进行充分验证。例如，对线程管理模块，测试了线程的创建、启动、挂起、恢复、删除等基本操作，以及不同调度策略下的线程切换和优先级管理；对内存管理模块，测试了内存的分配、释放、重分配等操作以及对标准 alloc 的容器支持；对定时器模块，测试了定时器的创建、启动、停止、重启等操作，以及定时器回调函数的执行等。

5.4. 集成测试

在模块间集成测试中，我们重点关注各模块协作及系统整体功能正确性，设计了以下典型测试场景：

1. 并发性测试

- 并发线程创建：多线程同时创建 RT-Thread 线程，验证线程管理安全性。
- 优先级调度验证：创建不同优先级线程并记录执行顺序，确保调度器优先级语义正确。
- 内存碎片化测试：模拟复杂内存分配释放场景，验证内存管理器健壮性。

2. 压力测试

- 大规模线程创建：创建 100 个线程同时运行，测试系统高负载稳定性。
- 内存分配压力测试：进行 1000 次随机大小内存分配释放，验证内存管理器性能。
- 定时器密集测试：创建 50 个定时器同时运行，测试时钟系统处理能力。

3. 边界条件和错误处理

- 边界值测试：测试最小/最大优先级、最小内存分配、零大小分配等边界情况。
- 错误场景模拟：测试重复启动线程、重复挂起线程、无效线程 ID 等异常处理。
- 资源耗尽测试：模拟内存不足等资源耗尽场景，验证系统错误处理机制。

4. 系统稳定性测试

- 长时间运行测试：验证系统长期稳定性。

5.5. 性能基准测试

为验证 RusT-Thread 的实时性能，我们设计了针对 RTOS 核心指标的性能基准测试体系。测试重点关注时间确定性和系统响应的可预测性，这是实时操作系统区别于通用操作系统的关键特征。我们选择了四项核心指标进行评估：中断延时测试验证系统对外部事件的响应速度，响应时间测试评估任务调度的实时性，上下文切换时间 测试衡量线程切换效率，线程启动时间测试检验系统资源分配性能。这些测试不仅帮助我们发现性能瓶颈和优化空间，更重要的是为系统的实时性保证提供了量化依据，确保 RusT-Thread 能够满足嵌入式和工业控制应用的严格时间要求[4] [5]。

5.5.1. 中断延时测试

我们利用 Cortex-M 内核自带的 SysTick (系统滴答定时器)来精确测量中断延时。SysTick 本质上是一个 24 位的硬件自减计数器。其工作原理如下：

- **计数与触发：**计数器以系统时钟频率从一个预设的重载值(Reload Value)开始递减。当计数值从 1 减到 0 时，硬件会立即触发 SysTick 中断。
- **自动重载：**在触发中断的同一时钟周期，硬件会自动将重载值重新加载到计数器中，使其无缝开始下一轮递减，整个过程无需任何软件干预。

由于中断服务程序的执行会存在微小的延迟，而 SysTick 计数器在此期间并未停止。因此，我们可以在中断服务程序的入口处，立即读取此刻 SysTick 计数器的当前值。通过这个差值，我们就能精确计算出

中断延时。计算公式为：

$$\text{中断延时} = (\text{重载值} - \text{当前值})/\text{系统时钟频率}$$

我们系统测得 1000 次平均中断延时为 1.21 us。

5.5.2. 响应时间测试

事件响应时间是指从事件发生到系统完成相应处理的总时间，包括事件检测、任务调度和执行处理逻辑的全过程。此指标衡量了系统对外部事件的端到端处理能力，是评估实时操作系统性能的综合指标。

在我们的测试中，使用随机数生成器模拟事件的随机生成，每隔相同时间生成一个随机数并与一个给定的概率值比较，当小于此概率值时，就生成一个事件，此事件的优先级也为随机数，可以证明，事件的间隔服从泊松分布。

我们将事件分为三种优先级：高中低，同时创建三个不同优先级的处理程序来处理事件，测量平均响应时间和各优先级的响应时间，结果如图 10 所示。

Figure 10. Event response time test results
图 10. 事件响应时间测试结果

一般硬实时操作系统响应时间的指标如表 1 所示。

Table 1. Hard real-time operating system response time indicators
表 1. 硬实时操作系统响应时间指标

优先级	高	中	低
响应时间(us)	1~5	5~20	20~100

可以看到我们三种优先级事件的响应时间均符合硬实时操作系统的要求。

5.5.3. 上下文切换时间测试

上下文切换时间是指操作系统从一个线程切换到另一个线程所需的时间，包括保存当前线程状态和恢复目标线程状态的过程。该指标对多任务系统的整体性能有显著影响，它决定了系统在任务间切换的效率。上下文切换时间越短，系统在高负载下的响应性越好，尤其在资源受限的嵌入式系统中，高效的上下文切换能显著提高处理器利用率和系统吞吐量。

我们测试了两个相同优先级线程来回切换 5000 次的平均时间，并执行了 100 次这样的测试，求得线程切换的时间性能如表 2 所示。

Table 2. Time performance of thread switching
表 2. 线程切换的时间性能

测试项目	结果
总测试次数	100
平均切换时间	1.75 μs
最小切换时间	1.11 μs
最大切换时间	2.58 μs
切换时间标准差	0.31 μs

5.5.4. 线程创建时间测试

线程创建时间是指从发起创建线程请求到新线程可以被调度执行所需的时间。该指标反映了系统动态资源分配和任务管理的效率。在需要频繁创建临时任务的应用场景中(如 Web 服务器、动态负载系统)，高效的线程创建机制可以显著降低系统开销，提高资源利用率。对于嵌入式实时系统，快速的线程创建能力也有助于系统在运行时更灵活地调整工作负载，适应变化的环境需求。

由于 flash 大小限制，单次测试创建 50 个线程的平均时间，再做 100 次测试求平均，得到测试结果如表 3 所示。

Table 3. Average thread creation time
表 3. 线程平均创建时间

测试项目	结果
总测试次数	100
平均创建时间	2.69 μs

6. RusT-Thread 与 RT-Thread 性能对比

RT-Thread 官方给出了他们的性能测试结果如图 11 [6]。

RT-Thread性能指标

RT-Thread 实时内核基于 Zynq7020平台测试的性能指标数据如下：

中断延时

系统心跳时钟频率 1000 Hz，测试总样本数 100000，99.59% 的中断延时数据分布在 280 ns 到360 ns 之间。

	最小值(ns)	平均值(ns)	最大值(ns)
中断延时时间	162	321	948

上下文切换指标

功能	最小值(us)	平均值(us)	最大值(us)
Thread switch by mailbox	0.801	0.854	1.399
Thread switch by semaphor	0.711	0.762	1.405
Thread switch by suspend	0.597	0.633	1.177

线程间通信指标

功能	最小值(us)	平均值(us)	最大值(us)
MBox send time	0.123	0.135	0.525
MBox receive time	0.135	0.136	0.228
MemPool allocate time	0.105	0.117	0.519
MemPool free time	0.093	0.099	0.249
Semaphore take time	0.099	0.111	0.222
Semaphore release time	0.093	0.099	0.138

线程创建指标

功能	最小值(us)	平均值(us)	最大值(us)
Thread create time	2.696	2.969	3.579
Thread init time	2.132	2.402	3.474

Figure 11. RT-Thread official performance test results
图 11. RT-Thread 官方性能测试结果

其测试基于的硬件平台是 Zynq 7020，该开发板的主频为 800 MHz，而我们使用的 QEMU 模拟的是 stm32f405 的开发板，主频为 168 MHz，在对比性能时应考虑相关的硬件资源。

Table 4. Performance comparison between RusT-Thread and RT-Thread
表 4. RusT-Thread 与 RT-Thread 性能对比

指标	RT-Thread	RusT-Thread	折合后的等效时间
中断延时(ns)	321	1210	254
上下文切换(us)	0.633	2.58	0.542
线程创建(us)	2.969	2.60	0.546

综上，我们重写后的 RusT-Thread 操作系统的实时性与 RT-Thread 相当甚至更加优秀(表 4)，这符合我们当初制定的性能提升的目标。

7. RusT-Thread 优点和缺点

7.1. 优点

1. 安全导向的设计语言。Rust 的所有权机制使得 RusT-Thread 操作系统在内存安全和并发安全上很有优势[7] [8]。

2. 高度模块化并且可以定制。在搭建 RusT-Thread 操作系统时，我们基于 Rust 语言特性，建立了可模块化的系统仓库，定义改写模块方便快捷。

3. 平台具备高度可扩展性，虽然目前仅支持 Cortex-M4 平台，但模块化的 Hardware 利于后续扩展平台。

4. 多样的算法特性选择。在线程调度上提供多种调度算法，如优先级、优先级 + RR 等；在内存管

理上支持伙伴系统、小内存分配器等多种内存分配方式。

5. 性能优秀。在 Rust 改写后的系统线程上下文切换性能与原 C 程序性能相当。

6. 测试点丰富。我们对各个模块开发了功能测试程序，利于后续的调试开发。

7.2. 缺点

1. 相关文档尚不完善。由于时间精力，我们暂未维护起完善的文档体系。

2. Rust 所有权机制等致使代码可读性差。为开发带来一定困难。

3. 外设周边尚未开发支持。

4. 异常反馈系统尚不完善，待后续开发。

8. 总结与展望

历经数月的攻坚克难，RusT-Thread 项目终于迎来了阶段性成果。我们成功地将 RT-Thread nano 内核的核心功能，包括线程调度、内存管理、中断处理、时钟服务、IPC 等，用 Rust 语言进行了高质量的重构与实现。这不仅是一次技术栈的迁移，更是在嵌入式实时操作系统领域，对 Rust 语言安全性、并发性和现代语言特性的一次深度实践与验证。

8.1. 核心成果：安全性与性能的双重提升

1. 内存安全基石

最大的收获莫过于 Rust 强大的所有权系统和借用检查机制带来的根本性改变。通过 RTIntrFreeCell 等创新设计，我们有效解决了嵌入式系统中全局共享数据访问这一高危痛点，将数据竞争、野指针、缓冲区溢出等 C 时代常见的“幽灵”从编译期就扼杀在摇篮里。内核服务的错误处理也因 Option/Result 变得更加健壮和可预测。

2. 性能不妥协

我们并非单纯追求“安全”而牺牲效率。精心优化的调度算法(如基于位图 + FFS 的优先级调度、创新的多级反馈队列)、高效的小内存分配器实现，以及 Rust 零成本抽象的特性，共同确保了 RusT-Thread 在 QEMU 模拟环境下的性能指标(中断延时、上下文切换、线程创建、响应时间)与原版 C 实现的 RT-Thread Nano 相当甚至略有优势。这证明了 Rust 完全有能力胜任对实时性要求苛刻的嵌入式场景。

3. 代码精简与清晰

Rust 的现代语言特性(如 trait、泛型、丰富的标准库容器)显著提升了代码的表达力和可维护性。最直观的体现是，在实现同等甚至更多功能(如更优的定时器二分查找算法)的前提下，RusT-Thread 的核心代码量(约 5500 行)相比原 C 版(约 9600 行)有了显著的精简。模块化设计和清晰的抽象也让代码结构更易于理解和扩展[9]。

4. 扎实的验证体系

我们构建了涵盖单元测试、集成测试和全面的性能基准测试(内存性能、中断延时、响应时间、上下文切换、线程创建)的验证体系。详实的数据不仅证明了系统的功能正确性，也为性能优化和后续迭代提供了坚实基础。

8.2. 挑战与突破：在“裸机”上驾驭 Rust

项目过程并非一帆风顺。调试手段匮乏时，我们深度依赖半宿主打印和内联汇编调试技巧；硬件对接和启动流程的复杂性，通过 cortex-m、cortex-m-rt 等库结合内联汇编巧妙化解；汇编与 Rust 联合编译

调试的难题，最终以内联汇编统一在 Rust 源码中的方案优化解决[8]；而困扰嵌入式开发的全局变量问题，则由 RTIntrFreeCell + lazy_static 的组合拳提供了安全可靠的 Rust 式解决方案。每一次挑战的克服，都加深了我们对 Rust 在嵌入式裸机环境应用的理解。

8.3. 展望未来：构建更强大、更开放的 RusT-Thread 生态

RusT-Thread 的诞生只是一个起点，我们对其未来充满期待：

1. 功能深化与扩展

(1) 丰富软件生态

系统性地移植 RT-Thread 社区成熟的核心软件包(网络协议栈 lwIP/PicoTCP、文件系统 LittleFS/SPIFFS、GUI 组件等)，是当务之急。我们将致力于构建标准化的 Rust-C 互操作层接口，让海量的现有 C 语言资源能更顺畅地融入 Rust 生态。

(2) 高级内核特性

实现更完善的内存管理策略(Slab, MemHeap)、支持更丰富的 IPC 机制(消息队列、邮箱、事件集)、探索多核(SMP)支持将是内核层面的重要方向。

(3) 人性化体验

当前错误处理主要透传底层错误码。未来计划引入结构化的 Rust-native 错误类型，并集成分级日志与堆栈追踪功能，让异常反馈更清晰、调试更高效。

2. 生态建设与普及

(1) 广泛的硬件支持

目前已在 Cortex-M4 上验证。下一步将适配更多主流架构如 Cortex- M0+/M3/M7 和 RISC-V，目标是覆盖更广泛的物联网和边缘计算硬件平台。

(2) 清晰的开发者体验

完善多级文档体系是生态繁荣的关键：

- 代码级：维护详尽的 rustdoc API 文档，包含示例和安全性说明。
- 模块级：编写硬件抽象层(HAL)指南、驱动移植教程、核心模块设计解析等。
- 入门级：提供面向应用开发者的、易于上手的使用手册和丰富的示例项目，显著降低 Rust 嵌入式开发的门槛。

(3) 工具链优化

持续优化代码体积(如替换重型打印宏)、提升构建体验，并探索更好的调试支持集成(如更深入的 GDB 支持)。

RusT-Thread 项目是一次勇敢的尝试，它证明了 Rust 在资源受限的实时操作系统领域不仅可行，更能带来显著的安全性和开发效率提升。我们重构的不仅是一套代码，更是在探索嵌入式系统开发的未来范式。代码已开源，这只是一个开始。我们热切期待更多对 Rust 和嵌入式系统感兴趣的开发者加入，共同打磨 RusT-Thread，将其打造成为一个真正安全、高效、易用的开源实时操作系统选择，为国产嵌入式基础软件生态注入新的活力！安全至上，性能无忧，Rust 让嵌入式未来更可期。

致　谢

本项目来自于中国科学技术大学操作系统 H 课大作业，在这里感谢任课老师邢凯对本项目的指导，感谢课程助教团队对本项目的帮助！

参考文献

[1] RT-Thread 文档中心[EB/OL]. https://www.rt-thread.org/document/site/#/, 2025-04-21.

[2] Klabnik, S. and Nichols, C. (2018) The Rust Programming Language. No Starch Press.

[3] 陈渝, 尹霞, 张峰. Rust 语言机制与安全性[C]//第 39 次全国计算机安全学术交流会论文集. 2024.

[4] 胡霜, 华保健, 欧阳婉容, 樊淇梁. 语言安全研究综述[J]. 信息安全学报, 2023,, 8(6): 64-83.

[5] Criterion 测试工具[EB/OL]. https://docs.rs/criterion/latest/criterion/, 2025-05-23.

[6] Pompeii, E. (2024) How to Benchmark Rust Code with Criterion. https://bencher.dev/learn/benchmarking/rust/criterion/

[7] 千锋教育第 3 章 RT-Thread 内核介绍[EB/OL]. https://zhuanlan.zhihu.com/p/641915283, 2025-04-21.

[8] 乐鑫科技 Rust + 嵌入式: 强力开发组合[EB/OL]. https://zhuanlan.zhihu.com/p/628575325, 2025-04-21.

[9] RT-Thread 产品性能[EB/OL]. https://www.rt-thread.com/products/Performance-38.html, 2025-04-21.

Call for Papers

Embedded Technology and Intelligent Systems

嵌入式技术与智能系统

国际中文期刊征文启事

https://www.hanspub.org/journal/etis

ISSN: 3065-1220

《嵌入式技术与智能系统》是一本开放获取、关注集成传统嵌入式技术与新兴智能系统的前沿研究最新进展的国际中文期刊，期刊特别注重软件算法、芯片设计与硬件实施的协同进展，以及理论研究与工程实践的紧密结合，面向学术界学者、产业界专家与工程师、学生及技术爱好者，关注中国领先产业集群的广阔发展潜力。本期刊强调发表原创性、创新性及具有实用价值的研究成果。 该期刊由汉斯出版社出版，全球发行，现诚邀相关领域的学者投稿。

投稿领域：

人工智能技术-边缘计算-端侧智能和大模型嵌入式应用
GPT-行业GPT以及GPT在嵌入式及智能系统研发中的应用
信息物理融合系统(CPS)-物联网技术-感知计算和无线传感网-泛在电力物联网-智能电表-储能技术-智能输变电
嵌入式系统结构-嵌入式操作系统与中间件-Linux、安卓和开源鸿蒙应用
实时操作系统-虚拟化和容器技术-混合关键系统
嵌入式软件形式化建模-软件测试和仿真-功能安全技术
嵌入式软件云原生技术-CI/CD和DevOpt-微服务
软硬件协同设计-开源指令集和开源芯片-RISC-V产业生态
嵌入式SoC技术--MCU 创新与生态-FPGA/DSP技术和应用
AI芯片和算法-存储技术-GPU 技术-视觉芯片及嵌入式显控应用
CAN和工业总线技术-时间敏感系统-电机控制-PLC和工业PC
无线通信技术-WiFi/蓝牙/Mesh/蜂窝/5G网络-物联网安全-低功耗设计
嵌入式系统课程改革-物联网和AI教学研究-职业教育-企业人才培养
嵌入式智能系统应用（智能家居、可穿戴设备、机器人、医疗电子、汽车电子和航空航天等）

征文要求及注意事项：

1. 稿件务求主题新颖、论点明确、论据可靠、数字准确、文字精炼、逻辑严谨、文字通顺，具有科学性、先进性和实用性；
2. 稿件必须为中文,且须加有英文标题、作者信息、摘要、关键词和规范的参考文献列表；
3. 稿件请采用WORD排版，包括所有的文字、表格、图表、附注及参考文献；
4. 从稿件成功投递之日起，在2个月内请勿重复投递至其他刊物。本刊不发表已公开发表过的论文。文章严禁抄袭，否则后果自负；
5. 本刊采用同行评审的方式，审稿周期一般为5~14日。

欲了解更多信息请登录 https://www.hanspub.org/journal/etis　　联系邮箱：etis@hanspub.org

Hans汉斯

嵌入式技术与智能系统

主编：何立民　北京航空航天大学教授

主办：汉斯出版社　珠海昊谷电子科技有限公司

编辑：《嵌入式技术与智能系统》编委会

网址：https://www.hanspub.org/journal/etis

电子邮箱：etis@hanspub.org

出版：汉斯出版社