嵌入式虚拟化技术探索及实践

原创 张云飞，吴春光 嵌入式系统专家之声 2025年11月8日 08:00

2025.11.08

本文字数：8858
预计阅读时长：23分钟

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

作者：张云飞[1]；吴春光[2]

DOI:10.12677/etis.2025.23013, PDF, HTML, XML

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。

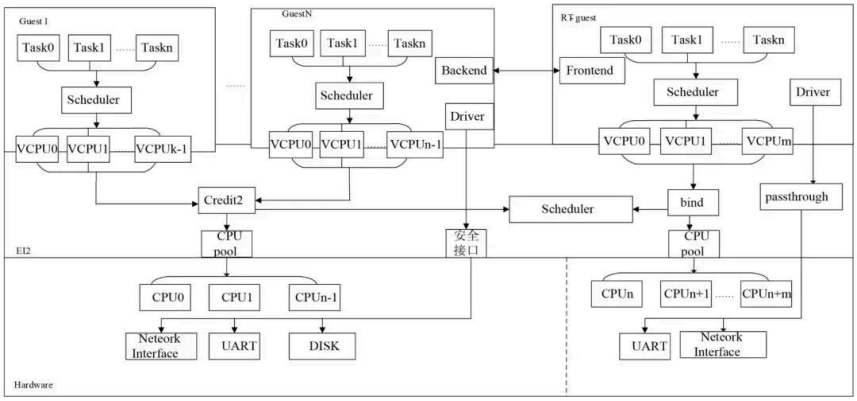


Figure 1. Diagram of the dynamic scheduling + static isolation

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。

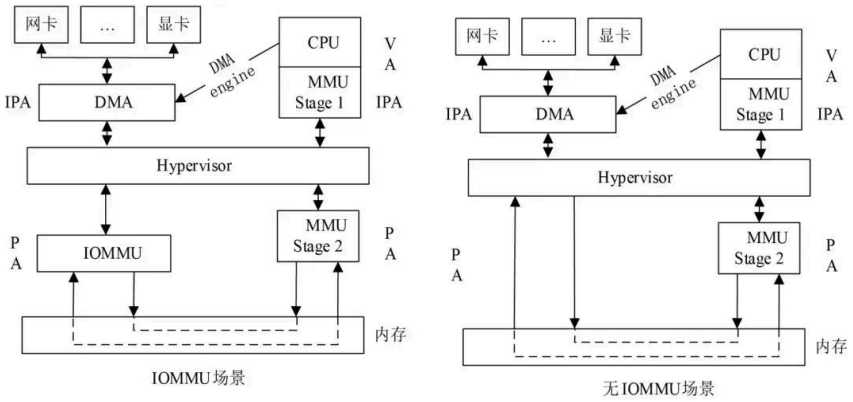


Figure 2. Diagram of memory pass-through under virtualization

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：

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

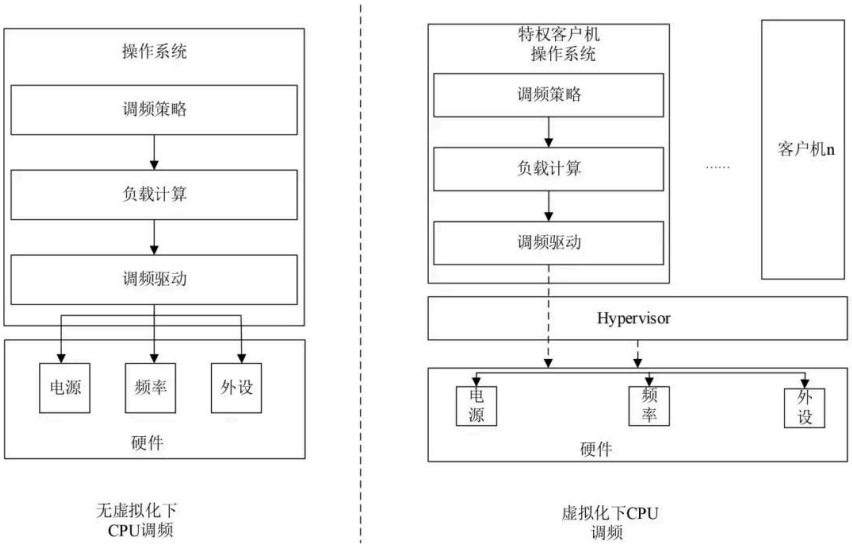


Figure 3. Diagram of CPU frequency modulation after virtualization

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 总结与展望

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*1aeowoq*_g cl_au*NTM4ODcyNjAwLjE3NjA5MzEyNTA.*FPAU*NTM4O

（作者单位：1.麒麟软件有限公司，天津；2.麒麟软件有限公司，北京）

□ □

本文由《嵌入式技术与智能系统》授权发表，原文刊登在2025年第3期。《嵌入式技术与智能系统》杂志由汉斯中文开源期刊学术交流平台出版，是一本关注传统嵌入式技术与新兴智能

系统前沿技术最新进展的国际中文期刊，编委团队汇聚了国内知名嵌入式系统专家与学者。
阅读原文了解期刊详情并可下载论文PDF版本。

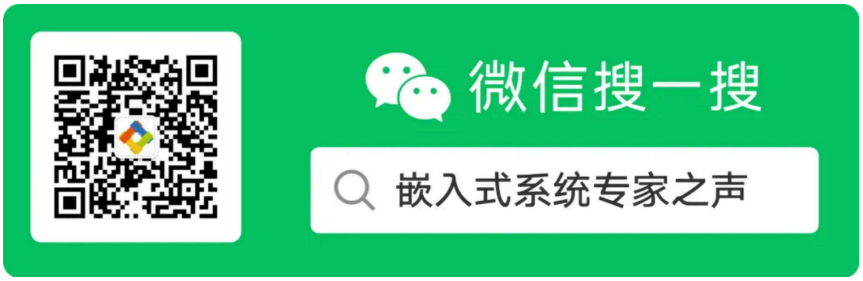
长按识别二维码投稿

欢迎高校师生和企业专家给我刊投稿

嵌入式系统专家之声推荐搜索

嵌入式系统 | 虚拟化 | 物联网

【点击上方👆搜索词条可查看号内更多其他内容】



关注我们，了解更多精彩内容

点击阅读原文

转发，点赞，在看，安排一下?

期刊论文 · 目录

上一篇 · 人工智能发展下的嵌入式系统教学与人才培养探索

阅读原文