# QEMU简介

QEMU是一款开源的模拟器、仿真器，完全的软件模拟，可以模拟出多种类型的CPU。**主要对单片机或者嵌入式常用的一些处理器模拟，嵌入式开发中使用广泛，比如arm，sparc，riscv等架构处理器。**

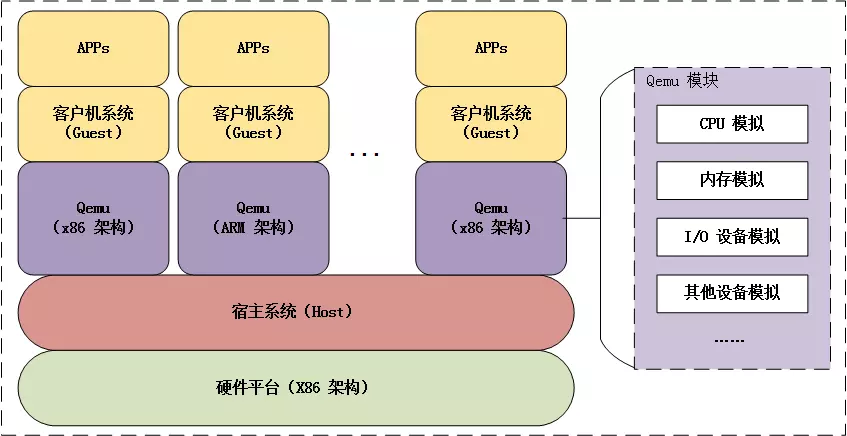
QEMU作为一个主机上的VMM(virtual machine monitor)，通过动态二进制转换来模拟CPU，并提供一系列的硬件模型，使guest os认为自己和硬件直接打交道，其实是同QEMU模拟出来的硬件打交道，QEMU再将这些指令翻译给真正硬件进行操作。通过这种模式，guest os可以和主机上的硬盘，网卡，CPU，CD-ROM，音频设备和USB设备进行交互。但由于所有指令都需要经过QEMU来翻译，因而性能会比较差，使用KVM模块进行加速。

KVM是硬件辅助的虚拟化技术，主要负责比较繁琐的CPU和内存虚拟化，而QEMU则负责I/O虚拟化，两者合作，相得益彰。

QEMU主要提供两种功能给用户使用：

* 用户态模拟器(User-mode emulation)，利用动态代码翻译机制来执行不同于主机架构的代码。能够在任意支持的架构上为另一个linux/BSD运行程序
* 全系统模拟(Full-system emulation)，利用其它VMM(Xen,KVM等)来使用硬件提供的虚拟化支持，创建接近于主机性能的虚拟机。能够在任意支持的架构上为任何机器运行一个完整的操作系统

Qemu是一个开源的托管虚拟机，通过纯软件来实现虚拟化模拟器，几乎可以模拟任何硬件设备。比如：Qemu可以模拟出一个ARM系统中的：CPU、内存、IO设备等，然后在这个模拟层之上，可以跑一台ARM虚拟机，这个ARM虚拟机认为自己在和硬件进行打交道，但实际上这些硬件都是Qemu模拟出来的。



当QEMU作为机器仿真器使用时，QEMU可以通过动态代码翻译机制(dynamic translation)在不同的机器上仿真任意一台机器(如ARM板)，并执行不同于主机架构的代码。同时由于动态代码翻译机制，它也能够实现不错的性能。

当QEMU作为虚拟器时，QEMU的优点在于其纯软件实现的虚拟化模拟器，几乎可以模拟任何硬件设备，但正因为Qemu是纯软件实现的，所有的指令都要经过它的转换，所以性能会严重降低。所以在生产环境中，大多数的做法都是配合KVM做加速来完成虚拟化工作，因为KVM是硬件辅助的[虚拟化技术](https://zhida.zhihu.com/search?q=%E8%99%9A%E6%8B%9F%E5%8C%96%E6%8A%80%E6%9C%AF&zhida_source=entity&is_preview=1)，主要负责比较繁琐的CPU和内存虚拟化，而Qemu则负责I/O虚拟化，两者合作各自发挥自身的优势，相得益彰。

## 虚拟化方式

QEMU提供两种方式的CPU虚拟化：基于KVM实现的和基于中间代码实现的

* 基于KVM实现：

全虚拟化是一种QEMU通过模拟完整计算机硬件的方式来运行客户机操作系统的虚拟化技术。QEMU使用KVM（内核虚拟机）来加速虚拟化。这种方法在没有硬件虚拟化支持的系统上可以实现虚拟化，但是其性能通常比较低。此类虚拟化能够支持不同类型的客户机操作系统，这需要处理器支持VMX功能。基于KVM的方式，直接使用host CPU执行target CPU的指令，性能接近host上的性能，但是需要target CPU和host CPU是相同的构架。

* 基于中间代码实现（TCG，Tiny Code Generator）：

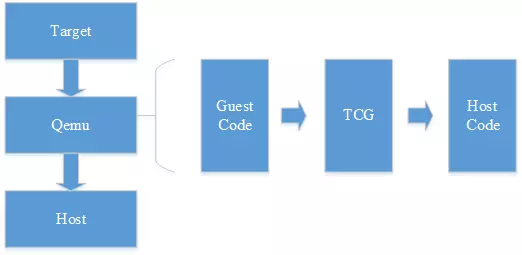
当处理器不支持虚拟化技术（VMX）时，QEMU使用全虚拟化技术来实现CPU虚拟化。在这种情况下，QEMU会完全模拟虚拟机的硬件环境，包括模拟CPU、内存、磁盘、网络和其它硬件设备等。QEMU通过软件来模拟CPU指令集，安装并运行客户机操作系统，当指令执行时，QEMU的CPU执行模拟程序可以在主机CPU上进行模拟。因此，在非虚拟化CPU上执行的指令被QEMU模拟并转换为可以在CPU上运行的指令。实现的具体方式时通过纯软的方式将目标架构代码转换为中间代码（相当于前端），然后将中间代码转换为宿主机架构的代码（相当于后端），这种方式虽然会造成仿真效率的降低，但是不需要强制CPU支持VMX。

### TCG

TCG(tiny code generator)，这种方式的基本思路是用纯软件的方式把targetCPU的指令先翻译成所谓的中间码，然后再把中间码翻译成host CPU 的指令，把targetCPU指令翻译成中间码的过程叫整个过程的前端，中间码翻译成hostCPU的过程对应的叫做后端。给qemu增加一个新CPU的模型需要既增加前端也增加后端，如果要把整个系统支持起来，还要增加基础设备以及user mode的支持。

为了移植性和通用性方面的考虑，qemu定义了micro-op，qemu代码翻译流程如下

target instruction->micro-op->tcg->host instruction



### KVM和TCG对比

在QEMU中，地址空间的处理方式受到不同CPU类型的影响，TCG和KVM的处理方式会有所不同。

对于TCG类型的CPU，因为它是在QEMU用户空间中运行的，所以可以允许多个虚拟CPU同时运行，每个虚拟CPU拥有各自的AddressSpace。在内存访问时，每个虚拟CPU会使用自己的AddressSpace映射虚拟地址到物理地址，不同虚拟CPU之间彼此独立、互不干扰。因此，在TCG模式下，一个CPU可以拥有多个地址空间。

相反，对于KVM类型的CPU，因为它是在Linux内核中运行的，和其它Linux进程一样，只能拥有一个AddressSpace。在内存访问时，KVM虚拟CPU使用的AddressSpace来自于它所运行的进程，即QEMU进程，所有虚拟CPU共享同一份AddressSpace映射表，每个虚拟CPU使用的是同一份映射关系，从而实现对模拟器内的地址空间的统一管理和访问。

所以，无论采用哪种CPU类型，AddressSpace在QEMU中都是非常重要的，因为它是QEMU模拟器内存管理的关键组成部分。在使用QEMU进行模拟时，需要根据具体需求和CPU类型等因素来选择使用相应的地址空间处理方式。

## QEMU设备对象模型-QOM

QEMU整个项目是C语言写的，但是QEMU处理的对象例如主板、CPU、总线、外设等实际上存在很多继承的关系。QEMU为了方便整个系统的构建，提供了一套面向对象编程模型—QOM(QEMU Object Module)，几乎所有设备如CPU、内存、总线等都利用QOM来实现。

* 各种架构CPU的模拟和实现

QEMU中要实现对各种CPU架构的模拟，而且对于一种架构的CPU，比如X86\_64架构的CPU，由于包含的特性不同，也会有不同的CPU模型。任何CPU中都有CPU通用的属性，同时也包含各自特有的属性。为了便于模拟这些CPU模型，面向对象模型必不可少。

* 模拟device和bus的关系

在主板上，一个device会通过bus与其他的device相连接，一个device上可以通过不同的bus端口连接到其他的device，而其他device也可以进一步通过bus与其他设备连接，同时一个bus上也可以连接多个device，这种device连bus、bus连device的关系，qemu需要模拟出来。为了方便模拟设备的这种特性，面向对象的编程模型也是必不可少的。

QOM是对QEMU中对象的一个抽象层，通过QOM可以对QEMU中的各种资源进行抽象、管理。比如设备模拟中的设备创建、配置和销毁。

## QEMU-ARM内核架构支持

* QEMU's TCG emulation includes support for the Armv5, Armv6, Armv7 and Armv8 versions of the A-profile architecture
* QEMU's TCG emulation support for R-profile CPUs is currently limited. We emulate only the Cortex-R5 and Cortex-R5F CPUs.
* QEMU's TCG emulation includes support for Armv6-M, Armv7-M, Armv8-M, and Armv8.1-M versions of the M-profile architucture.

# QEMU and Systemc

两种方案：

* QEMU-SC，QEMU和systemc model作为两种单独的simulator
* QBOX，把QEMU作为一个TLM wrapper嵌入到systemc主导的simulator

## QEMU-SC

图示

描述已自动生成

图示

描述已自动生成

图示

描述已自动生成

图示

描述已自动生成

## QBOX

图示

描述已自动生成

图示

描述已自动生成

图示

描述已自动生成

图示

描述已自动生成

图示

描述已自动生成