(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 119201357 A
(43)申请公布日 2024.12.27

(21)申请号 202411485103.1

(22)申请日 2024.10.23

(71)申请人 上海兆芯集成电路股份有限公司
地址 200120 上海市浦东新区张江高科技园区金科路2537号301室

(72)发明人 李用维 宋永峰 丁迎来 石鑫

(74)专利代理机构 北京林达刘知识产权代理事务所(普通合伙) 11277
专利代理师 刘新宇 宋晓雯

(51)Int.Cl.
G06F 9/455(2018.01)
G06F 11/26(2006.01)
G06F 11/22(2006.01)

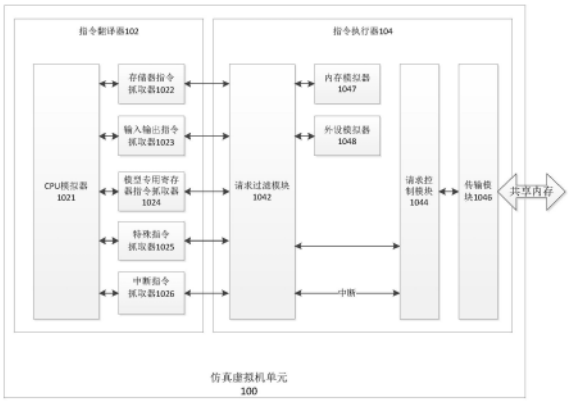
权利要求书2页 说明书14页 附图5页

(54)发明名称

计算机系统

(57)摘要

本公开涉及计算机系统,包括仿真虚拟机,其包含将仿真虚拟机指令翻译成测试平台能执行的指令的翻译器和指令执行器,仿真虚拟机利用共享内存将指令发送到测试平台;指令翻译器包括CPU模拟器,模拟器与BIOS、与OS进行指令的接收和发送,指令翻译器还包括多个从模拟器抓取相应指令的指令抓取器;指令执行器包括:请求过滤模块对从指令翻译器接收到的各类指令过滤和分发,将被过滤剩余的测试平台执行测试所需要的指令发送到请求控制模块;请求控制模块将被过滤剩余的指令发送给传输模块,并从传输模块接收来自测试平台的测试执行结果;传输模块利用共享内存将测试平台所需要的指令发送给测试平台。



CN 119201357 A

1.一种计算机系统,所述计算机系统包括仿真虚拟机单元,被配置为利用所述仿真虚拟机单元运行基本输入输出系统、操作系统和应用软件,其特征在于,

所述仿真虚拟机单元包括被配置为将仿真虚拟机指令翻译成测试平台能够执行的指令的指令翻译器和指令执行器,所述仿真虚拟机单元利用共享内存将指令发送到所述测试平台上执行,并接收来自所述测试平台的测试执行结果;

所述指令翻译器包括CPU模拟器,所述CPU模拟器与所述基本输入输出系统以及与所述操作系统进行所述指令的接收和发送,

所述指令翻译器还包括多个从所述CPU模拟器抓取相应指令的指令抓取器;以及

所述指令执行器包括:请求过滤模块、内存模拟器、外设模拟器、请求控制模块和传输模块,其中,

所述请求过滤模块被配置为对从所述指令翻译器接收到的各类型指令进行过滤和分发,将被过滤剩余的所述测试平台执行测试所需要的指令发送到所述请求控制模块,其中,被过滤剩余的指令包括中断指令;

所述内存模拟器和所述外设模拟器被配置为从所述请求过滤模块接收被过滤出的指令,并向所述请求过滤模块返回所述内存模拟器和所述外设模拟器各自的处理结果;

所述请求控制模块被配置为将所述被过滤剩余的指令发送给所述传输模块,并从所述传输模块接收来自所述测试平台的测试执行结果;以及

所述传输模块被配置为利用共享内存将所述测试平台执行测试所需要的指令发送给所述测试平台,并接收所述测试平台的测试执行结果。

2.根据权利要求1所述的系统,其中,各所述指令抓取器包括:存储器指令抓取器、输入输出指令抓取器、模型专用寄存器指令抓取器、特殊指令抓取器和中断指令抓取器;

所述各类型指令包括:存储器指令、输入输出指令、模型专用寄存器指令、特殊指令和中断指令。

3.根据权利要求1所述的系统,其中,各所述指令抓取器根据仿真虚拟机指令各自的特性来抓取与各所述指令抓取器相对应的指令代码。

4.根据权利要求1所述的系统,其中,所述请求过滤模块根据如下任意之一对指令进行过滤:

根据所述测试平台上的待测设计的类型对指令进行过滤;或者

根据指令所访问的地址区间来对指令进行过滤;或者

根据所述测试平台上的待测设计对于指令的需求,对获取自所述指令翻译器的指令进行过滤。

5.根据权利要求4所述的系统,其中,所述请求过滤模块根据所述地址区间所对应的是内存、外设还是待测设计,来判断所述指令是否为所述待测设计所需要的指令。

6.根据权利要求1所述的系统,其中,所述请求控制模块对被过滤剩余的指令进行编码、追踪和流量控制,并且将从所述测试平台上的待测设计返回到所述共享内存的数据回填到所述仿真虚拟机单元、再由所述仿真虚拟机单元返回给发出所述指令的所述基本输入输出系统或者操作系统。

7.根据权利要求6所述的系统,其中,所述请求控制模块将期望有数据返回的指令按照识别编码进行记录,并且响应于从所述传输模块接收到所述待测设计返回的数据,根据所

述识别编码从所述记录中获取指令信息,并根据所获取的指令信息将所述返回的数据发送给所述仿真虚拟机单元中的所述CPU模拟器。

8.根据权利要求2所述的系统,其中,所述传输模块对所接收到的输入输出指令、模型专用寄存器指令和特殊指令中的至少一者进行打包,从而在一个数据包中以与各指令相对应的格式来传送输入输出指令、模型专用寄存器指令和特殊指令中的至少一者。

9.根据权利要求1所述的系统,其中,所述传输模块监听所述共享内存的通信信道,所述通信信道包括中断传输信道以及存储器指令、输入输出指令和模型专用寄存器指令的读取数据信道;

所述传输模块响应于监听到所述读取数据信道上的数据,获取所述数据并将获取到的数据发送给所述CPU模拟器、所述内存模拟器、或者连接在所述请求过滤模块和所述请求控制模块之间的高速缓存模块;

所述传输模块还响应于监听到所述中断传输信道上的中断,将中断上报给所述指令翻译器中的中断指令抓取器。

10.根据权利要求1所述的系统,其中,仿真虚拟机进程和测试平台进程通过共享内存队列通信协议来共享所述计算机系统的物理内存,

所述传输模块将不同类型的指令或数据分别置入不同的共享内存队列中,使得所述测试平台并发地从队列中获取到指令或数据。

11.根据权利要求10所述的系统,其中,所述共享内存队列通信协议的消息格式包括共享内存报头,所述共享内存报头包括用于在所述仿真虚拟机单元和所述测试平台之间进行信息交换的边带信息,所述边带信息在初始化时由所述仿真虚拟机单元设置并在传输所述中断指令时使用。

12.根据权利要求11所述的系统,其中,所述共享内存报头包括在初始化时由所述仿真虚拟机单元设置的与所述测试平台之间的通信信道的数量和描述信息,使得所述测试平台根据所述数量和所述描述信息来绑定所述仿真虚拟机单元所创建的通信信道。

13.根据权利要求1所述的系统,其中,所述测试平台通过共享内存与所述仿真虚拟机单元进行通信,将与出现故障的待调试软件系统相同的软件系统在所述测试平台中运行以进行故障复现,并且响应于所述测试平台复现出所述故障,利用调试手段来确定所述故障的故障类型,其中,所述故障类型包括:软件故障、硬件故障。

14.根据权利要求13所述的系统,其中,所述测试平台上的待测设计为芯片或者子系统级设备,响应于所述芯片或者子系统级设备在测试平台上运行待调试软件系统时出现故障,将所述待测设计改变为更小规模的子系统级设备或模块级设备,并将与出现故障的待调试软件系统相同的软件系统在所述更小规模的子系统级设备或模块级设备中运行以进行故障复现调试。

15.根据权利要求1所述的系统,其中,所述测试平台包括连接到待测设计的信号切换器,从测试开始直到所述测试平台的环境初始化结束为止,所述信号切换器将所述待测设计连接到所述测试平台的原有环境以对所述待测设计进行初始化。

计算机系统

技术领域

[0001] 本公开涉及计算机领域,尤其涉及计算机系统对软硬件指令的仿真。

背景技术

[0002] 随着云计算、大数据、个人计算机等领域对于处理器的算力增长需求越来越高,处理器的复杂度变得越来越大,且要求产品的迭代速度越来越快。而对处理器产品、尤其是芯片验证的时间往往占据整个芯片研发时间的70%以上。传统的验证方法在验证的前期后期分别进行硬件和软件的验证,并且软件系统验证依赖于需要昂贵板卡资源的硬件仿真系统(emulator),导致验证效率和质量较低。

发明内容

[0003] 尽管计算机系统软硬件各自的研发时间紧任务重,但本领域一直缺乏全周期的软硬件协同验证。如何提高芯片的验证效率和质量、进而缩短处理器研发时间成为了关注的问题。

[0004] 有鉴于此,本公开提出了一种计算机系统,所述计算机系统包括仿真虚拟机单元,被配置为利用所述仿真虚拟机单元运行基本输入输出系统、操作系统和应用软件,所述仿真虚拟机单元包括被配置为将仿真虚拟机指令翻译成测试平台能够执行的指令的指令翻译器和指令执行器,所述仿真虚拟机单元利用共享内存将指令发送到所述测试平台上执行,并接收来自所述测试平台的测试执行结果;所述指令翻译器包括CPU模拟器,所述CPU模拟器与所述基本输入输出系统以及与所述操作系统进行所述指令的接收和发送,所述指令翻译器还包括多个用于从所述CPU模拟器抓取相应指令的指令抓取器;以及所述指令执行器包括:请求过滤模块、内存模拟器、外设模拟器、请求控制模块和传输模块,其中,所述请求过滤模块被配置为对从所述指令翻译器接收到的各类型指令进行过滤和分发,将被过滤剩余的所述测试平台执行测试所需要的指令发送到所述请求控制模块,其中,被过滤剩余的指令包括中断指令;所述内存模拟器和所述外设模拟器被配置为从所述请求过滤模块接收被过滤出的指令,并向所述请求过滤模块返回所述内存模拟器和所述外设模拟器各自的处理结果;所述请求控制模块被配置为将所述被过滤剩余的指令发送给所述传输模块,并从所述传输模块接收来自所述测试平台的测试执行结果;以及所述传输模块被配置为利用共享内存将所述测试平台执行测试所需要的指令发送给所述测试平台,并接收测试平台的测试执行结果。

[0005] 利用上述的全指令抓取器、请求过滤器,根据本公开的各方面可以实现以下效果中的至少一方面:大大提高了仿真系统的通用性,能够减少仿真所依赖的仿真器板卡等硬件资源;通过仿真虚拟机与测试平台的交互,可以实现软硬件协同测试,从而缩短芯片验证所耗时长;可通过调整请求过滤模块的指令过滤规则,适配不同待测设计的测试,从而实现灵活测试。

[0006] 根据下面参考附图对示例性实施例的详细说明,本公开的其它特征及方面将变得

清楚。

附图说明

[0007] 包含在说明书中并且构成说明书的一部分的附图与说明书一起示出了本公开的示例性实施例、特征和方面,并且用于解释本公开的原理。

[0008] 图1示出根据本公开一实施例的计算机系统所包括的仿真虚拟机单元的示意框图。

[0009] 图2示出根据本公开一实施例的共享内存与物理内存的关系的示意图。

[0010] 图3示出根据本公开一实施例的利用共享内存队列在仿真虚拟机单元和测试平台之间进行通信的示意框图。

[0011] 图4示出根据本公开一实施例的非并发仿真与并发仿真所用时间的对比示意图。

[0012] 图5示出根据本公开一实施例的仿真虚拟机单元利用测试平台进行测试的结构示意框图。

[0013] 图6示出根据本公开一实施例的测试中所用测试平台结构的例示图。

[0014] 图7示出基于测试的原有环境快速搭建混合仿真方案的示意流程图。

[0015] 图8示出根据本公开一实施例的可以适用的多种待测设计的示意框图。

具体实施方式

[0016] 以下将参考附图详细说明本公开的各种示例性实施例、特征和方面。附图中相同的附图标记表示功能相同或相似的元件。尽管在附图中示出了实施例的各种方面,但是除非特别指出,不必按比例绘制附图。

[0017] 在这里专用的词“示例性”意为“用作例子、实施例或说明性”。这里作为“示例性”所说明的任何实施例不必解释为优于或好于其它实施例。

[0018] 另外,为了更好的说明本公开,在下文的具体实施方式中给出了众多的具体细节。本领域技术人员应当理解,没有某些具体细节,本公开同样可以实施。在一些实例中,对于本领域技术人员熟知的方法、手段、元件和电路未作详细描述,以便于凸显本公开的主旨。

[0019] 图1示出根据本公开一实施例的计算机系统所包括的仿真虚拟机单元100的示意框图。

[0020] 计算机系统包括仿真虚拟机单元100,被配置为利用仿真虚拟机单元100运行基本输入输出系统(BIOS)、操作系统和应用软件。如图1所示,仿真虚拟机单元100包括被配置为将仿真虚拟机指令翻译成测试平台能够执行的指令的指令翻译器102和指令执行器104,仿真虚拟机单元100利用共享内存将指令发送到所述测试平台上执行,并接收来自所述测试平台的测试执行结果。指令翻译器102包括CPU模拟器1021,CPU模拟器1021与基本输入输出系统以及与操作系统进行所述指令的接收和发送,指令翻译器102还包括多个指令抓取器。在一个具体实现中,例如可以包括:用于从CPU模拟器1021抓取相应指令的存储器指令抓取器1022、输入输出(IO)指令抓取器1023、模型专用寄存器(MSR)指令抓取器1024、特殊(Special)指令抓取器1025和中断指令抓取器1026。指令执行器104包括:请求过滤模块1042、内存模拟器1047、外设模拟器1048、请求控制模块1044和传输模块104。其中,请求过滤模块1042被配置为对从指令翻译器102接收到的各类型指令(例如,在一个具体实现中可

以包括存储器指令、输入输出指令、模型专用寄存器指令、特殊指令和中断指令)进行过滤和分发,将被过滤剩余的所述测试平台执行测试所需要的指令发送到请求控制模块1044,其中,被过滤剩余的指令包括中断指令;内存模拟器1047和外设模拟器1048被配置为从请求过滤模块1042接收被过滤出的指令,并向请求过滤模块1042返回内存模拟器1047和外设模拟器1048各自的处理结果;请求控制模块1044被配置为将所述被过滤剩余的指令发送给传输模块1046,并从传输模块1046接收来自所述测试平台的测试执行结果;以及传输模块1046被配置为利用共享内存将所述测试平台执行测试所需要的指令发送给所述测试平台,并接收测试平台的测试执行结果。

[0021] 本发明人首先认识到,所用的测试平台上的待测设计可以是单个模块,也可以是子系统等。由于待测设计的功能具有单一性,可能不执行全部的指令,且不同待测设计能够执行的指令集合也可能是不同的。可以在仿真虚拟机中截获待测设计能够执行的指令发给待测设计处理,并接收待测设计返回的结果,而由仿真虚拟机上的虚拟设备来执行待测设计所不能执行的指令(即被过滤掉的指令),从而实现本公开的通用性,仿真虚拟机上的虚拟设备可以是例如仿真虚拟机等的开源虚拟机上默认存在的,也可以是利用仿真虚拟机框架而开发的。

[0022] 本发明人通过对传统虚拟机进行研究和二次开发,从中获取软件执行过程中产生的全部指令,筛选出待测设计可以执行的指令并与待测设计进行混合仿真。同时,还在虚拟机中修改或增加了相关虚拟设备,从而执行被过滤出的、待测设计所不能执行的指令。由此,系统软件或应用软件无需修改即可以在本公开的计算机系统的仿真虚拟机中执行,降低了验证过程中额外工作量,同时也为在不同平台上对比创造了条件。

[0023] 在仿真虚拟机单元100内部:CPU模拟器1021会解析软件产生的指令,可以通过内嵌源代码的方式获取IO、MSR(model specific register,模型专用寄存器)、定制的指令;内存模拟器1047实现了MMU(Memory Management Unit),并且通过内嵌源代码的方式获取memory和MMIO指令。外设模拟器1048负责模拟各个外设设备的行为。可以通过内嵌源代码的方式获取到设备相关的指令。

[0024] 在一个可能的实现中,指令抓取器1022～1026可以根据不同的仿真虚拟机指令各自的特性,从源代码中抓取与各类型指令相对应的指令代码。

[0025] 在一个可能的实现中,请求过滤模块1042可以根据测试平台上的待测设计(DUT)对于指令的需求,对获取自指令翻译器102的指令进行过滤,其中,被过滤出的指令在仿真虚拟机单元100内的内存模拟器1047和外设模拟器1048上执行,所述待测设计执行被过滤剩余的指令。

[0026] 在一个可能的实现中,请求过滤模块1042根据如下任意之一对指令进行过滤:根据所述测试平台上的待测设计的类型;或者根据指令所访问的地址区间;或者根据所述测试平台上的待测设计对于指令的需求,对获取自所述指令翻译器的指令进行过滤。

[0027] 在一个可能的实现中,请求过滤模块1042可以根据所述地址区间所对应的是内存、外设还是待测设计,来判断所述指令是否为所述待测设计所需要的指令。

[0028] 在一个可能的实现中,请求控制模块1044可以对被过滤剩余的指令进行编码、追踪和流量控制,并且将从待测设计返回到共享内存的数据回填到仿真虚拟机单元100、再由仿真虚拟机单元100返回给发出所述指令的基本输入输出系统或者操作系统。

[0029] 在一个具体实现中,请求控制模块1044可以将期望有数据返回的指令按照识别编码(ID)进行记录,并且响应于从传输模块1046接收到待测设计返回的数据,根据ID从所述记录中获取指令信息,并根据所获取的指令信息将返回的数据发送给仿真虚拟机单元100中的CPU模拟器1021。

[0030] 在一个可能的实现中,传输模块1046可以对所接收到的输入输出指令、模型专用寄存器指令和特殊指令中的至少一者进行打包,从而在一个数据包中以与各指令相对应的格式(例如,3种格式)来传送输入输出指令、模型专用寄存器指令和特殊指令中的至少一者。

[0031] 在一个可能的实现中,传输模块1046可以监听共享内存的通信信道。通信信道可以包括中断传输信道以及存储器指令、输入输出指令和模型专用寄存器指令的读取数据信道。

[0032] 例如,传输模块1046响应于监听到读取数据信道上的数据,获取所述数据并将获取到的数据发送给CPU模拟器1021、内存模拟器1047、或者连接在请求过滤模块1042和请求控制模块1044之间的高速缓存(cache)模块;

[0033] 再例如,传输模块1046还响应于监听到中断传输信道上的中断,将中断上报给指令翻译器102中的中断指令抓取器1026。

[0034] 本公开一实施例的特点之一是实现了全指令的仿真。需要说明的是根据不同仿真需求,外设模拟器1048模拟的外设可以有所不同,这里仅进行示意说明。在外设模拟器1048中,可以利用仿真虚拟机现有的常用外设、例如USB无线电数据记录器(USDR),也可以自行开发其他外设设备。在一个具体实现中,在仿真虚拟机中的CPU模拟器1021、内存模拟器1047、外设模拟器1048中可以进行相应指令的处理。

[0035] 一个典型的计算机系统可以包括如下表所示的几种指令。

[0036]

指令类型	含义
存储器(Memory)指令	用来访问动态随机存取存储器(DRAM)或者存储器管理输入输出(MMIO)寄存器，Memory 指令中包含 MMIO 指令
输入输出(IO)指令	用来访问一些特殊的寄存器
模型专用寄存器(MSR)指令	用来访问 MSR 寄存器
特殊(Special)指令	一些特殊用途的循环，用来向硬件发送一些特殊信号
定制指令	可以根据指令格式自定义一个新指令，配合仿真虚拟机完成一些自定义的操作

[0037] 相对于现有技术,本发明人进一步开发了至少以下部件/功能:针对全指令集中各类型指令的抓取、请求过滤模块、中断处理、请求控制模块、传输模块、及相应的共享内存通信协议。根据本公开一实施例,上述各部件/模块的功能例如详述如下。

[0038] 1)虚拟机部分

[0039] 在本公开实施例的一个具体实现中,在仿真虚拟机上运行基本输入输出系统、操作系统及应用软件。这些软件在虚拟机中运行产生各种各样的虚拟机指令,包括上表中的各种指令。

[0040] 2)仿真虚拟机的指令翻译器

[0041] 指令翻译器可以把各种类型的虚拟机指令翻译成宿主机可以执行的指令。在这个过程中,可以给每一类指令分别用一个抓取器(hooker)抓取上述表格中的各种指令,具体如下表:

[0042]

Hooker 名字	作用
存储器指令 hooker	抓取 memory 指令，并接收数据返回
输入输出指令 hooker	抓取 IO 指令，并接收数据返回
模型专用寄存器指令 hooker	抓取 MSR 指令，并接收数据返回
特殊指令 hooker	抓取 special 指令，并接收数据返回
定制指令 hooker	抓取定制指令，并接收数据返回

[0043] 在一个具体实现中,各指令抓取器根据例如虚拟机指令各自的特性来抓取相应的指令。例如,对于存储器指令抓取器,其可以基于内存模拟器中的内存管理,从代码中准确地截取转发来抓取存储器指令。需要说明的是,在仿真虚拟机中对各种类型的指令进行抓取这一构思在本领域从未被提出过,且在实现层面也并非本领域技术人员显而易见能容易想到的,而是需要对仿真虚拟机架构和虚拟机代码实现层级的非常深入的理解。

[0044] 3)指令执行器

[0045] QEMU完成指令翻译之后,就可以执行指令了。执行到抓取的测试平台所需要的指令时,可以按照例如下表的方式处理各类指令:

[0046]

指令	处理方式
Memory 指令	指令执行器还可以具有高速缓存(cache)模块，当 memory 指令的 memory 属性解析为可高速缓存(cacheable)属性，就访问 cache 模块，再由 cache 模块产生 cacheable 指令发给请求控制模块。如果是不可高速缓存(un-cacheable)属性，则直接发送指令给请求控制模块。
IO 指令	直接发送指令给请求控制模块。
MSR 指令	直接发送指令给请求控制模块。
Special 指令	直接发送指令给请求控制模块。
定制指令	直接发送指令给请求控制模块。(其中定制指令可用来进行功能定制或者新指令功能的模拟)。

[0047] 在一个具体实施方式中,向请求控制模块发送测试平台所需的输入输出指令、模型专用寄存器指令和特殊指令时,可以定义与三种指令各自相对应的格式,并利用指令执行器的传输模块对指令进行打包,以在例如一个数据包中以三种格式来传送这三种指令。

[0048] 另外,如图所示,本公开一个实施例的指令翻译器可以包括中断指令抓取器。中断指令抓取器能够实时监听请求控制模块是否有中断发来,如果有则把中断上报给仿真虚拟机的中断模块,然后通知CPU执行中断。

[0049] 4)请求过滤模块

[0050] 请求过滤模块用于对指令抓取器抓取并发送的各指令进行过滤、分发。具体地,请求过滤模块可以过滤出要由待测设计执行的指令发送至请求控制模块,并且过滤出由仿真虚拟机内部执行的指令,且依据指令的类型将其分发至内存模拟器和外设模拟器之一。

[0051] 具体地,memory指令可以分两种:一种是访问内存的memory指令,这类型指令通过请求过滤模块过滤后发给内存模拟器;另外一种是用于访问外设的存储器管理输入输出(MMIO),这类指令通过请求过滤模块过滤后发给外设模拟器。在实际测试时,当某些访问内存的memory指令需通过待测设计测试时,需将这部分指令由请求过滤模块过滤后经过请求控制模块和传输模块发送给待测设计,而对于其他无需通过待测设计测试的访问内存的memory指令则通过请求过滤模块过滤后发送给内存模拟器。同理,当某些MMIO需通过待测设计测试时,需将这部分指令由请求过滤模块过滤后经过请求控制模块和传输模块发送给待测设计,而对于其他无需通过待测设计测试的MMIO则通过请求过滤模块过滤后发送给外设模拟器执行。上述仅是具体说明了memory指令的筛选模拟,对于IO指令、MSR指令、Special指令、定制指令也是同样的筛选原则,对于这些类型中一个或多个具体指令需要通过待测试设计测试时,则通过请求过滤模块过滤后经过请求控制模块和传输模块发送给待测设计,不需要通过待测试设计测试的具体指令则发送到外设模拟器。

[0052] 5)请求控制模块

[0053] 请求控制模块控制传输模块对指令进行转发,以便在测试平台上的待测设计中执行指令。在一个具体实现中,可以从测试平台上的待测设计读取执行结果数据。请求控制模块等待传输模块从待测设计处获取到真正的执行结果数据,并将执行结果数据回填到仿真虚拟机的指令执行器,就如同指令是在仿真虚拟机的指令执行器自身进行处理执行的一样。

[0054] 在一个具体实现中,请求控制模块可以进行如下操作中的至少一者:

[0055] ◆调用传输模块来将数据包发送给测试平台。

[0056] ◆对于期望有数据返回的指令,比如存储器读(memory read),会对这种指令进行记录。当收到传输模块返回的数据之后,根据ID从该记录中获取指令信息,然后把该指令的数据返回给CPU模拟器,由CPU返回给对应的指令发出方(BIOS或者操作系统)。

[0057] ◆能够实时监听传输模块是否有中断发来,如果有则把中断上报给指令翻译器中的中断hooker,然后通知CPU模拟器执行中断。

[0058] 6)传输模块

[0059] ◆用来把指令打包,并根据类型把指令发送给共享内存队列的不同传输信道。

[0060] ◆用来监听测试平台用于将数据发送给仿真虚拟机的几个信道(共享内存队列),包括:监听memory/IO/MSR read数据信道,如果有memory/IO/MSR read数据会先统一发送给请求控制模块,由请求控制模块根据read请求的来源不同把数据发给不同的对象:如果read请求来源为高速缓存的memory read数据则发给高速缓存;如果read请求来源为CPU的memory read、IO read、MSR read数据则发给CPU;传输模块还监听中断传输信道,如果有中

断(包括高级可编程中断控制器(APIC)、系统管理中断(SMI)、非屏蔽中断(NMI)等中断则上报给指令翻译器中的中断指令抓取器。

[0061] 7)共享内存

[0062] 物理内存中的共享内存空间用来承载实现共享内存队列。共享内存是仿真虚拟机和测试平台之间传输信道的具体承载者。

[0063] 图2示出根据本公开一实施例的共享内存与物理内存200的关系的示意图。如图2所示,在物理内存200中的仿真虚拟机进程与测试平台进程所占用的内存空间之间,设置共享内存空间。通过共享内存,可以使仿真虚拟机进程和测试平台进程能够共享使用同一段物理内存。所有的通信可以通过向共享内存读写来完成的。这种直接访问内存的方式,具有速度快、可靠性高、实时性强的特点。能够把仿真虚拟机单元100的请求和数据快速传递到测试平台中。测试平台中待测设计的数据也能快速传递给仿真虚拟机单元100。

[0064] 图3示出根据本公开一实施例的利用共享内存队列在仿真虚拟机单元和测试平台之间进行通信的示意框图。在一个具体实现中,图3中的绿色部分可以运行快速仿真虚拟机进程(QEMU进程)。

[0065] 如图3所示,可以基于共享内存进程通信技术,在共享内存中实现独立的队列对仿真中不同类型的指令和数据进行独立并行的传输。在一个具体实现中,仿真虚拟机向测试平台发送的指令或数据通过下行队列传递,测试平台向仿真虚拟机发送的指令或数据通过上行队列传递。

[0066] 在一个具体实现中,共享内存可以由操作系统在物理内存上分配空间,且由仿真虚拟机进程和测试平台进程共享这部分物理内存空间。因此,测试平台可以直接从共享内存中获取到仿真虚拟机发下来的指令,就像测试平台在访问自己的队列一样。同理,仿真虚拟机就像从自己的队列中直接获取一样获得测试平台向仿真虚拟机发送的不同类型的指令或者数据。这种方式的优点包括以下至少之一:1)仿真速度快,因为共享内存速度是所有进程通信速度最快的;2)通信是可靠的,因为信息是直接在内存中受控传递的,通常不会出现丢包、数据错误等问题;3)仿真虚拟机和测试平台之间低耦合,仿真虚拟机和测试平台只需要操作队列即可,不用关心对方的实现机制;4)减少了数据转换,仿真虚拟机和测试平台的数据包可以直接使用,不必做协议转换;5)使用简单且可扩展性强,可以封装成应用程序接口(API);6)具有高并发的特点,具体说明见下文。

[0067] 在本公开一实施例的一个可能的实现中,仿真虚拟机进程和测试平台进程通过共享内存队列通信协议来共享计算机系统的物理内存。传输模块1046将不同类型的指令或数据分别置入不同的共享内存队列中,使得测试平台能够并发地从队列中获取到指令或数据。

[0068] 在本公开一实施例的一个具体实现中,共享内存队列通信协议的消息格式可以包括共享内存报头,所述共享内存报头包括用于在仿真虚拟机单元100和所述测试平台之间进行信息交换的边带信息,所述边带信息在初始化时由仿真虚拟机单元100设置并能够在传输所述中断指令时使用。

[0069] 例如,共享内存报头可以包括在初始化时由仿真虚拟机单元100设置的与测试平台之间的通信信道的数量和描述信息,使得测试平台能够根据所述数量和所述描述信息来绑定仿真虚拟机单元100所创建的通信信道。

[0070] 以下详细说明通信协议的消息格式。

[0071] 1)共享内存报头(header)

[0072] 共享内存报头可以存储一些状态信息,用于仿真虚拟机和模拟器或仿真器的测试平台之间进行通信状态、传输信道的数量和描述信息、以及进行和一些边带的信息交换。共享内存报头可以在初始化时由仿真虚拟机设置完成。在一个具体实现中,设置可以包括:创建共享内存,填写传输信道的数量、描述信息以及边带信息。共享内存报头信息设置完成后,所设置的信息对于仿真虚拟机、测试平台进程均是可知的。其中,边带信息在例如用户传输中断信号时使用。

[0073] 在一个具体实现中,仿真虚拟机在创建传输信道时,每创建一个传输信道,就把报头中传输信道的数量加1、并在报头中增加新创建的传输信道的描述信息。测试平台进程在与仿真虚拟机进行通信之前,会通过报头信息获取到传输信道的数量和描述信息,并根据这些信息绑定仿真虚拟机创建的传输信道。这样,仿真虚拟机和测试平台之间就可以通过传输信号进行通信了。

[0074] 共享内存报头的示例代码如下。

[0075]
```
typedef struct
{
    bool init_ok;
    bool is_running;
    key_t shm_key;
    bool closed;
    int channel_num;
    ChannelInfo channel_info[MAX_CHANNEL_NUM];
    int shm_pos;
    int shm_size;
    uint32_t 边带_intr;
    sem_t sem_mutex_边带;
} TpHeader;
```

[0076] 各个字段含义如下:

[0077]

名称	作用
init_ok	用来表明通信链路是否初始化成功
is_running	用来表明当前链路是否在工作状态
shm_key	共享内存的键值,是共享内存的唯一 ID

[0078]

closed	用来表明当前链路已经处于关闭状态
channel_num	指示通信链路中信道的个数
channel_info	记录每个信道的详细信息
shm_pos	共享内存池的偏移量
shm_size	共享内存池的总容量大小
sideband_intr	用来记录 testbench 中断引脚电平信息，这里通过共享内存中的 sideband_intr 变量来记录
sem_mutex_sideband	保证在多进程或多线程情况下能正确读写该变量，在共享内存中设置了一个互斥量 sem_mutex_sideband 用来对 sideband_intr 变量的读写进行加锁。(该互斥量是为中断信号服务的，通过该互斥量可保证 testbench 在写变量的时候 QEMU 不会读，同理，在 QEMU 读中断信号时 testbench 不会写中断信号。)

[0079] 2)共享内存队列元素

[0080] 通过共享内存队列作为通信信道。用C/C++的结构体作为共享内存队列传递的基本元素,因为结构体中元素都是可以定制的,所以信道传递的信息类型是支持定制化的。比如下面是一种memory read或write request传递所使用的结构体定义：

[0081]
```
typedef struct __attribute__((packed))
{
    unsigned char data[64];
    unsigned long be;
    unsigned int size;
    unsigned long addr;
    unsigned char core_id;
    unsigned int req_id;
    unsigned char type;
    unsigned char attr;
    unsigned char smm;
    unsigned char lock;
} RW_PACKAGE;
```

[0082] 在一个具体实现中,共享内存队列可以以例如三种结构来实现:多线程发送多线程接收、多线程发送单线程接收、单线程发送单线程接收。

[0083] 根据本公开一实施例,开发了共享内存队列通信协议,以实现仿真虚拟机单元100和测试平台之间的指令和数据的并发传输和高速仿真。

[0084] 图4示出根据本公开一实施例的非并发仿真与并发仿真所用时间的对比示意图。

[0085] 高并发原理如图4所示,其中图4的(a)展现的是传统非并发仿真方式,不同类型的请求处于同一个队列中。因此,队列前面的请求处理完毕之前,后边的请求会被阻塞无法被处理,(a)中示例总共需要花费6个时钟周期才能仿真结束,这是一种串行的方式。而图4的

(b)展现的是本公开的并发仿真方式,将不同类型的请求分别置于独立的队列中,可以并发从队列中取出指令进行仿真而不会互相阻塞。在(b)的具体示例中最终仅需花费3个时钟周期即可处理完成。由此,利用共享内存能够满足待测设计的时序要求,使得不同类型的请求在待测设计侧可以同步处理。

[0086] 图5示出根据本公开一实施例的仿真虚拟机单元利用测试平台进行测试的结构示意框图。如图5所示,顶层即Top层,Top层中例化使得测试平台504上运行DUT的各子模块(DUT、COSIM适配器以及DPI接口配置信息等),以及各子模块间连接关系信息。图5所示的是自顶向下搭建的混合仿真方案。仿真虚拟机单元502可以是开源的虚拟机。在一个具体实现中,可以采用快速仿真虚拟机(Quick EMUlator,QEMU)。仿真虚拟机单元502能够运行与真机相同的基本输入输出系统(BIOS)、操作系统及应用软件。

[0087] 仿真虚拟机单元502可以提供软件可以直接执行的平台,获取软件运行中产生的各类指令从而与测试平台(testbench)上的待测设计交互,并模拟所需的外设设备以使得软件能够顺利执行,而几乎不用额外适配。

[0088] 根据本公开一实施例的计算机系统既可以应用于协同模拟(co-simulation)系统也可以应用于协同仿真(co-emulation)系统,因为例如仿真虚拟机、共享内存、直接编程接口、协同适配器在两种系统上都是通用的。在协同模拟系统中,仿真虚拟机、共享内存、测试平台都可以运行在模拟服务器上。在协同仿真系统中,软件部分的仿真虚拟机、共享内存、直接编程接口的例如C语言部分可以运行在仿真服务器上,硬件部分的协同适配器、直接编程接口的硬件接口可以运行在仿真设备(即emulator)上。

[0089] 测试平台上的直接编程接口(DPI)可以与共享内存队列直接交互。在一个具体实现中,测试平台可以进行以协同模拟(COSIM)为主体的验证,利用直接编程接口把指令或者数据通过协同模拟适配器转换成接口时序信号与待测设计连接。在测试平台的顶层,可以例化使得待测设计运行的各子模块(例如待测设计、COSIM适配器以及直接编程接口配置信息等)、以及各子模块间的连接关系信息。

[0090] 测试平台所验证的软件可以是虚拟机上所运行的应用软件、基本输入输出系统、操作系统中的至少部分,所验证的硬件可以是例如通过硬件编程语言编译的verilog文件等经仿真后生成的待测设计。待测设计与上述软件协同测试。

[0091] 图6示出根据本公开一实施例的测试中所用测试平台结构的例示图。

[0092] 根据本公开的一个具体实现,在搭建计算机系统的测试平台时,可以针对待测设计从头搭建测试环境的顶层,对待测设计进行例化并对其初始化,最后连接到仿真虚拟机单元以完成例如协同模拟COSIM环境。这种从头搭建方式的优点是可以去除有些不用的模块,使得测试环境更加简洁,进而减少对模拟器或者仿真器资源的消耗或占用。

[0093] 根据本公开的一个实施方式,可以基于仿真测试的原有环境来快速搭建基于本公开的混合仿真的方案。原有环境可以指验证团队先前所搭建的测试平台的原有验证环境。在后续例如搭建协同模拟COSIM平台时,可以将原有环境中对待测设计进行初始化的部分继承进来,以缩短COSIM测试环境开发时间。

[0094] 图6所示为基于原有环境快速搭建混合仿真的方案。左侧与图1左侧部分相同,右侧在原有环境的基础上增加了一个信号切换器,在仿真开始至环境初始化结束期间,配置完成(cfg done)信号为0,信号切换器连通原有外围环境对DUT进行初始化。在环境初始化

结束之后,cfg done信号为1,信号切换器连通COSIM适配器进行后续的混合仿真。这种信号切换的做法,充分利用了原有环境复杂的初始化逻辑,把重点工作放在了混合仿真身上,减少额外的debug工作,在缩短搭建COSIM环境时间的同时也能保证环境的稳定性。适用于快速搭建COSIM环境,是一种方法学,能广泛应用于各个系统当中。

[0095] 图7示出基于测试的原有环境快速搭建混合仿真方案的示意流程图,对这一方法的流程进行了图示。

[0096] 在本公开实施例的一个具体实现中,测试平台可以提供包括待测设计的仿真验证环境。根据本公开,可以运行Windows、Linux等任意操作系统,执行任意开源或闭源应用软件,且无需对仿真虚拟机的软件进行修改。由于本公开的实施例具备至少如下两个特点:1)通用性,既可用于仿真虚拟机和模拟器搭建协同模拟(co-simulation)系统,又可用于仿真虚拟机和仿真器搭建协同仿真(co-emulation)系统;2)全系统仿真,软件可以不用修改就在如下各种测试平台上切换运行,因此在方法学上具备对比验证的优势,能够增加调试(debug)手段和灵活性,进而提高仿真验证效率。

[0097] 图8示出根据本公开一实施例的可以适用的多种待测设计的示意框图。下表对上述几种系统进行总结:

[0098]

平台编号	应用场景	软件可跨平台通用
图 8 的左上图	模块或小规模子系统 co-simulation，灵活且无需 emulator，例如待测设计可以为动态随机存取存储器控制器	是
图 8 的左下图	大规模子系统 co-emulation，速度快，使用 emulator 资源较少，例如：待测设计可以为驱动验证(DV)	是
图 8 的右上图	芯片仿真(例如 Full chip emulator 验证)，主流做法，依赖较多 emulator 资源	是
图 8 的右下图	真实硅片制成后进行验证或交付生成环境，debug 困难	是

[0099] 下面描述如何进行对比验证:

[0100] ●当软件系统(包括应用软件、BIOS、OS,在图1中也有展示)在图8右下图硅后验证或者生产系统中运行,系统出现故障(bug),这种bug既可能是软件bug也可能是硬件bug。由于在硅后系统debug一般比较困难,因此可以把相同的软件系统直接跨平台在图8的左下和右上图所示的系统中运行进行bug复现。当图8左下图中系统复现出问题后,可利用如前文所述的各种调试(debug)手段,包括波形、log、debug工具等,先确定是软件bug还是硬件bug,如果确定是软件bug,则协同软件开发团队,利用图8左下图的系统进一步debug,比如增加日志(log)、利用gdb工具调试等方式确定问题,最后由软件开发团队解决问题。如果确定是硬件bug,则协同硬件开发团队,利用图8左下图的系统进一步debug,比如增加log、波形等方式确定问题,最后由硬件开发团队解决问题。因为图8左下图的有前文所述的各种丰富debug手段,对帮助定位和解决问题有较大帮助。

[0101] ●当在图8右上图中遇到问题之后,也可以把相同的软件系统跨平台在图8左下图的所示系统中进行复现和解决,此时图8左下图的的优点有:1)占用emulator资源较少;2)

debug手段比图8右上图丰富。

[0102] ●在进行对比验证的一个具体实现中,当确定或怀疑问题是由特定的某个模块或小规模子系统引起时,则可以搭建图左上图所示的协同模拟COSIM系统。利用仿真模拟机和模拟器组建模块或小规模子系统的协同模拟系统,在不占用仿真器emulator的昂贵板卡资源的情况下,可以使与出现问题的待调试软件系统相同的软件系统跨平台运行在COSIM系统中,在调试中利用丰富的定位手段进行问题的定位和解决。

[0103] ●多个平台共同debug,不受限于某一个平台上特定因素的干扰,为debug提供了更多可能。比如:可以进行波形、log等的信息对比,利用QEMU的虚拟设备替代DUT中的模块进行对比运行,再分析波形和log,帮助确定问题等。

[0104] 当问题明确后,反馈给软件团队和硬件团队进行问题的修改,保证产品研发顺利进行。

[0105] 在一个可能的实现中,测试平台可以通过共享内存与仿真虚拟机单元100进行通信,将与出现故障的待调试软件系统相同的软件系统在所述测试平台中运行以进行故障复现,并且响应于所述测试平台复现出所述故障,利用调试手段来确定所述故障的类型。例如,故障类型可以包括软件故障、硬件故障等。

[0106] 在一个具体实现中,测试平台上的待测设计可以为芯片(full chip)或者子系统级设备,响应于芯片或子系统级设备在测试平台上运行待调试软件系统时出现故障,将待测设计改变为更小规模的子系统级设备或模块级设备,并将与出现故障的待调试软件系统相同的软件系统在所述更小规模的子系统级设备或模块级设备中运行以进行故障复现调试。

[0107] 在一个可能的实现中,测试平台包括连接到待测设计的信号切换器,从测试开始直到所述测试平台的环境初始化结束为止,所述信号切换器将所述待测设计连接到所述测试平台的原有环境以对所述待测设计进行初始化。

[0108] 根据本公开一实施例,可以对各种待测设计进行混合仿真,比如:动态随机存取存储器控制器(DRAMC)模块、外围组件快速互连(PCIE)模块、芯片互联子系统、或者作为较大单元的输入输出裸片(IO Die)系统等等。

[0109] 作为具体应用举例,以下将以对处理器芯片进行一次软件验证、一次硬件验证为例,对整个验证流程进行说明。

[0110] 以DRAMC(DRAM控制器)验证为例,流程可以包括如下步骤:

[0111] S1:硬件设计团队开发DRAMC design,release一个基础版本。

[0112] S2:硬件验证团队基于设计团队release的版本,搭建验证环境,release一个可运行的基础版本

[0113] S3:COSIM团队基于硬件验证团队release的验证环境,搭建仿真虚拟机-DRAMC co-simulation或者co-emulation环境,并进行软件验证、软硬件交互验证。

[0114] S4:当发现问题后,需要定位问题。在DUT侧可以抓取(dump)波形或者输出log。在一个具体实现中,测试平台可以运行在EDA厂商的工具里,常见EDA厂商均提供上述抓取工具。在仿真虚拟机侧可以通过输出log(包含软件自身打印的信息,和软件与仿真虚拟机交互的请求和数据)查看执行的请求和获取的数据,也可以通过GDBgnu开源调试器/调试工具动态获取当前程序执行的状态或查看变量值等方式进行问题的定位,因为仿真虚拟机可以

运行各种各样的软件,所以还可以利用其它软件工具进行问题的定位;在仿真虚拟机和DUT之间的通信通道上,即共享内存实现函数和DPI接口函数中,也有打印log。可以把DUT、仿真虚拟机、通信通道这三部产生的上述debug信息结合起来定位问题。比如通过波形查看到是DRAMC的寄存器没有正确配置导致工作不正常,此时可以通过传输通道的log确定软件是否正确下发配置请求,如果软件正确下发了,则定位问题可能在DUT中,需要设计团队检查波形和design。如果软件没有正确下发,则定位问题可能出现在软件一侧,需要软件开发团队通过仿真虚拟机输出的log或利用gdb等工具进行debug,如此反复,最终解决问题。

[0115] S5:硬件验证团队发现design bug,报告给硬件设计团队更新design,并同步更新验证环境,COSIM团队更新COSIM环境进行回归验证。

[0116] S6:COSIM团队发现design bug,报告给硬件验证团队和硬件设计团队进行确认,然后由硬件设计团队修改bug之后,交给COSIM团队和硬件验证团队回归验证。

[0117] S7:COSIM团队发现软件bug,提交给软件开发团队进行问题的确定和debug,然后由COSIM团队进行回归验证。

[0118] S8:S5-S7步骤同时进行,并且不断进行直到软件和design满足验证要求。

[0119] 大体上,本公开的应用场景可以包括如下至少之一:

[0120] √软件提前开发和验证。

[0121] √硬件提前进行软件验证或系统级验证。

[0122] √提前进行软硬件联调,包括系统软件bring up等。

[0123] √软硬件性能分析和调优工作。

[0124] √对比验证来调试芯片内部问题。

[0125] 在实际测试中,使用传统测试方式时,8核心设备需要消耗12个仿真器板卡,1核心设备需要消耗5个仿真器板卡。与此相对,根据本公开一实施例的测试方式,8核心和1核心设备测试均只需要消耗2个仿真器板卡,显著降低了资源消耗,提高了验证效率。由于本公开的各个方面所提供的计算机系统用于仿真的通用性,能够在芯片验证各个阶段、尤其是早期阶段,进行不同模块、子系统、系统级的软件混合模拟和协同模拟(co-simulation)、硬件混合仿真和协同仿真(co-emulation),并且允许软件不做任何修改就能与全芯片仿真(full chip emulation)、硅片(silicon)仿真进行对比验证、充分验证,提高验证效率和质量,有助于缩短产品上市时间。

[0126] 本公开提出的通用的全指令软硬件混合方法,其通用性体现在:可以在芯片研发周期的各个阶段,对任意模块、子系统、系统,利用无需修改的Windows、Linux等操作系统和BIOS、应用软件进行软件级/系统级的验证。全指令主要体现在对系统中软件产生的任意指令进行获取并通过协议传递给待测设计进行验证。共享内存队列通信协议具备并发能力强、仿真速度快、低耦合、通信可靠、减少数据转换、使用简单、可扩展性强的特点。本公开可以广泛应用于各个项目不同层级、不同阶段的软硬件协同并行验证,尤其是提前验证、软件系统验证,是一种验证方法学的延伸,能够提高验证效率和质量。

[0127] 以上已经描述了本公开的各实施例,上述说明是示例性的,并非穷尽性的,并且也不限于所披露的各实施例。在不偏离所说明的各实施例的范围和精神的情况下,对于本技术领域的普通技术人员来说许多修改和变更都是显而易见的。本文中所用术语的选择,旨在最好地解释各实施例的原理、实际应用或对市场中的技术改进,或者使本技术领域的其

它普通技术人员能理解本文披露的各实施例。

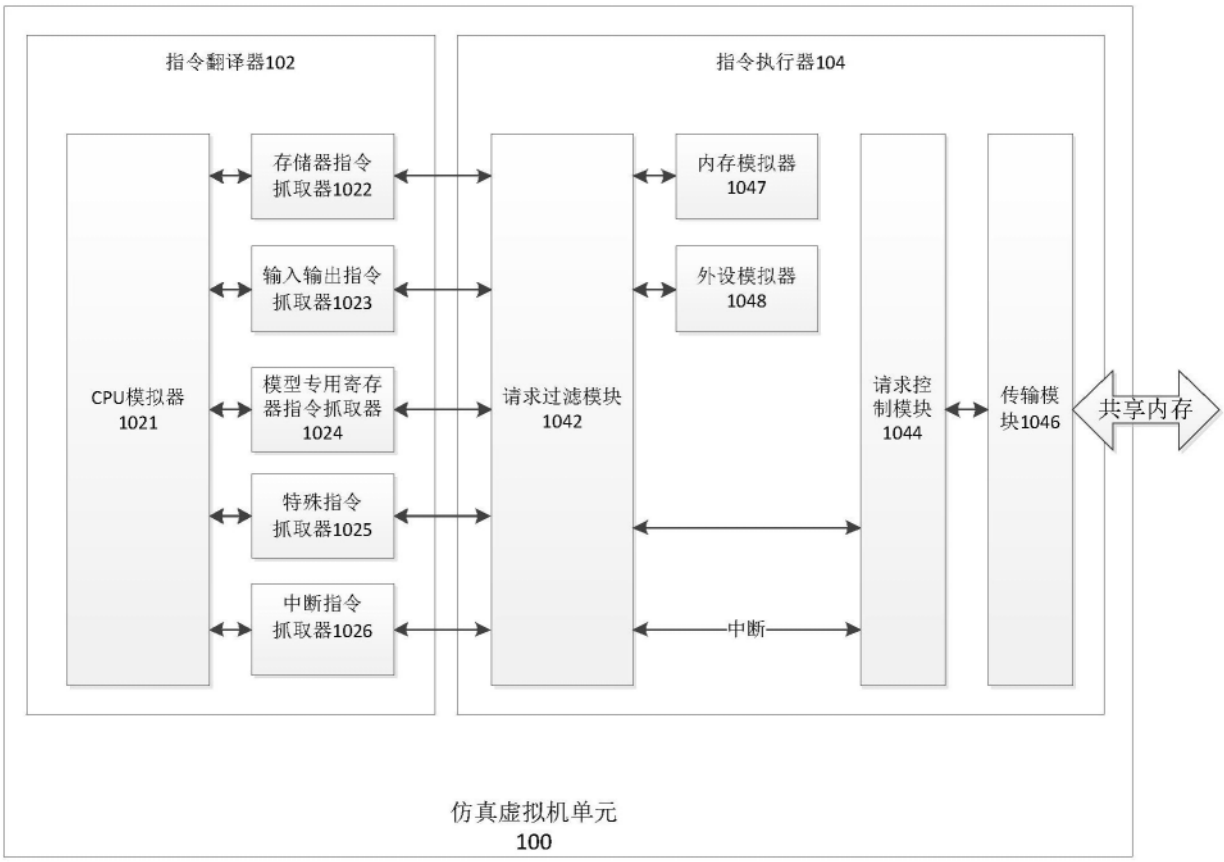


图1

物理内存
200

图2

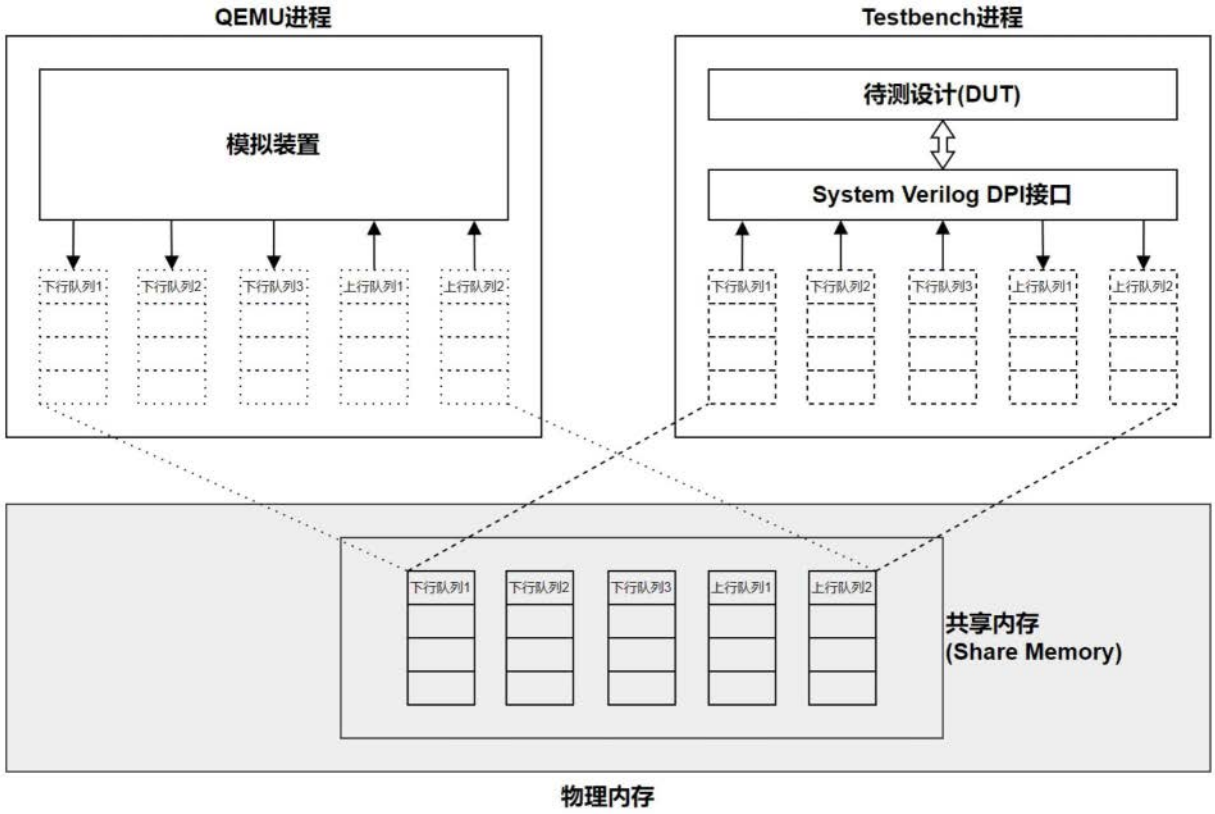
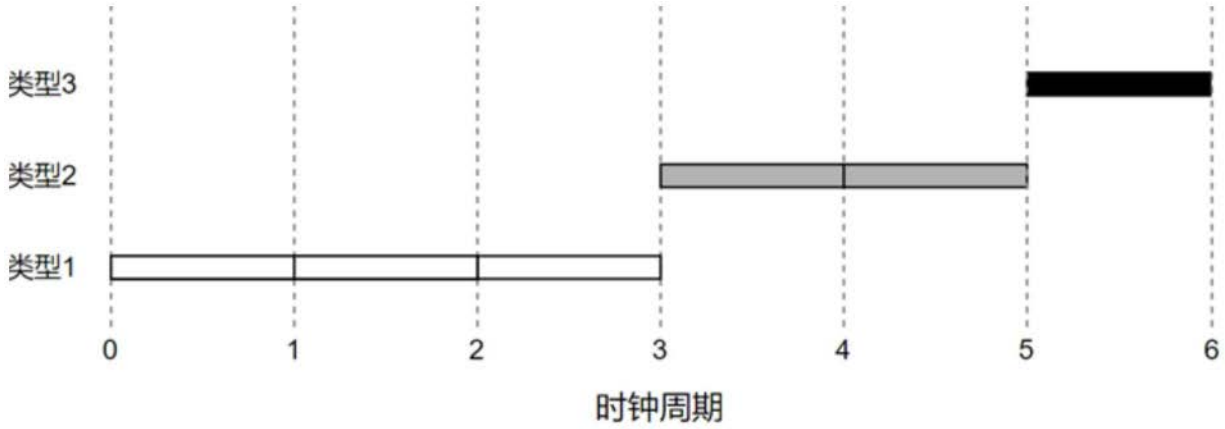
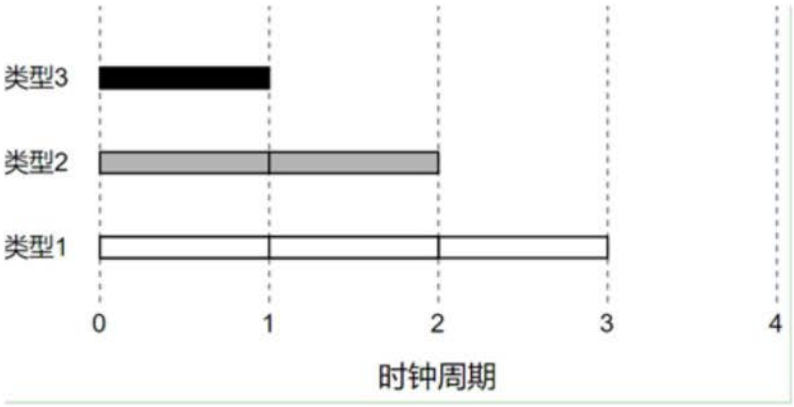

QEMU进程
模拟装置
下行队列1
下行队列2
下行队列3
上行队列1
上行队列2
Testbench进程
待测设计(DUT)
System Verilog DPI接口
下行队列1
下行队列2
下行队列3
上行队列1
上行队列2
下行队列1
下行队列2
下行队列3
上行队列1
上行队列2
共享内存
(Share Memory)
物理内存

图3



(a) 非并发仿真方式



(b)并发仿真方式

图4

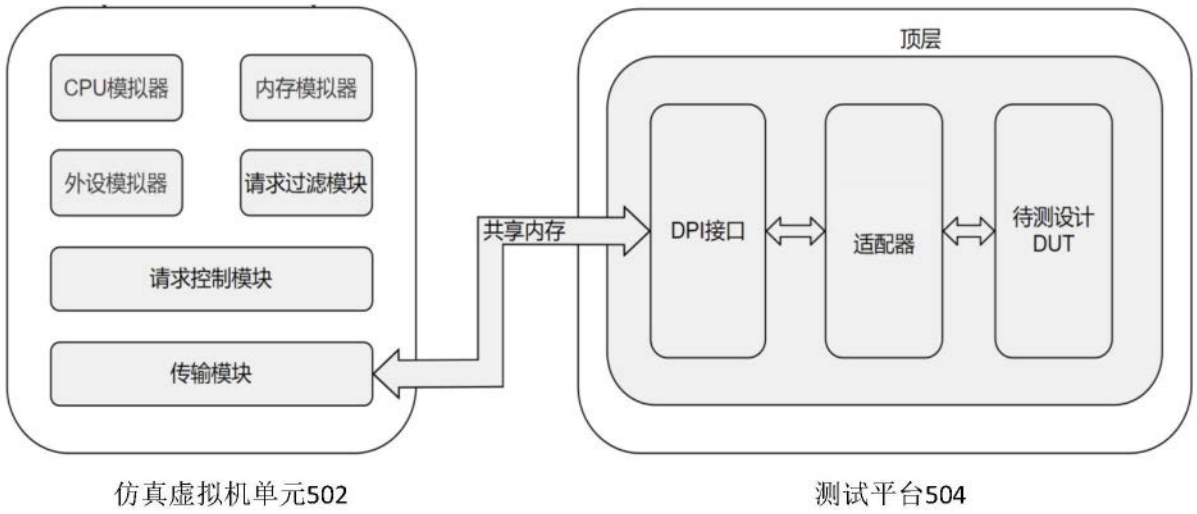

CPU模拟器
内存模拟器
外设模拟器
请求过滤模块
请求控制模块
传输模块
共享内存
顶层
DPI接口
适配器
待测设计
DUT
仿真虚拟机单元502
测试平台504

图5

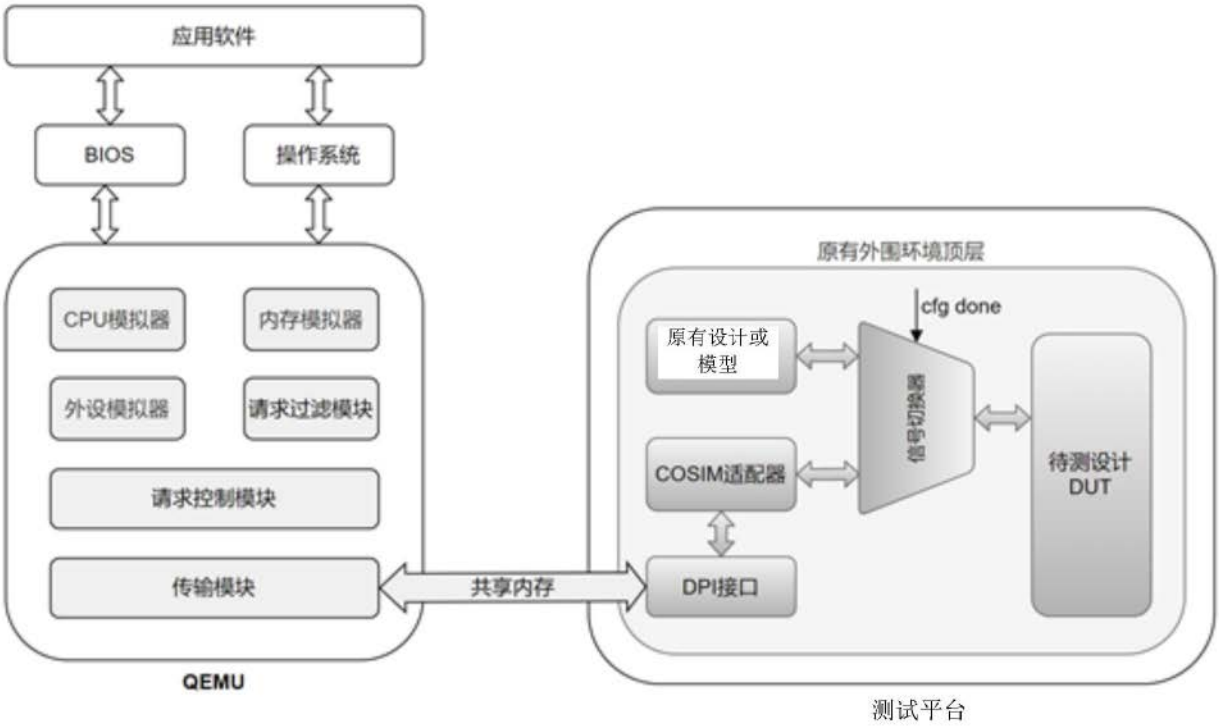

应用软件
BIOS
操作系统
CPU模拟器
内存模拟器
外设模拟器
请求过滤模块
请求控制模块
传输模块
QEMU
共享内存
原有外围环境顶层
cfg done
原有设计或
模型
COSIM适配器
信号切换器
DPI接口
待测设计
DUT
测试平台

图6

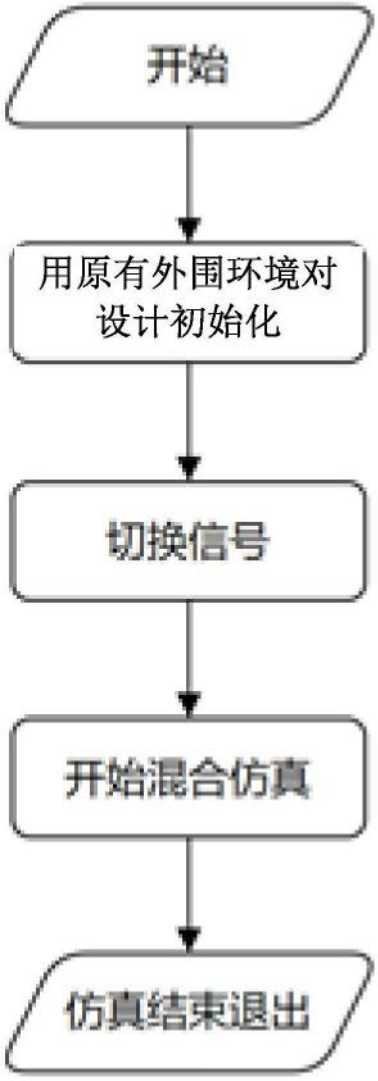

开始
用原有外围环境对设计初始化
切换信号
开始混合仿真
仿真结束退出

图7

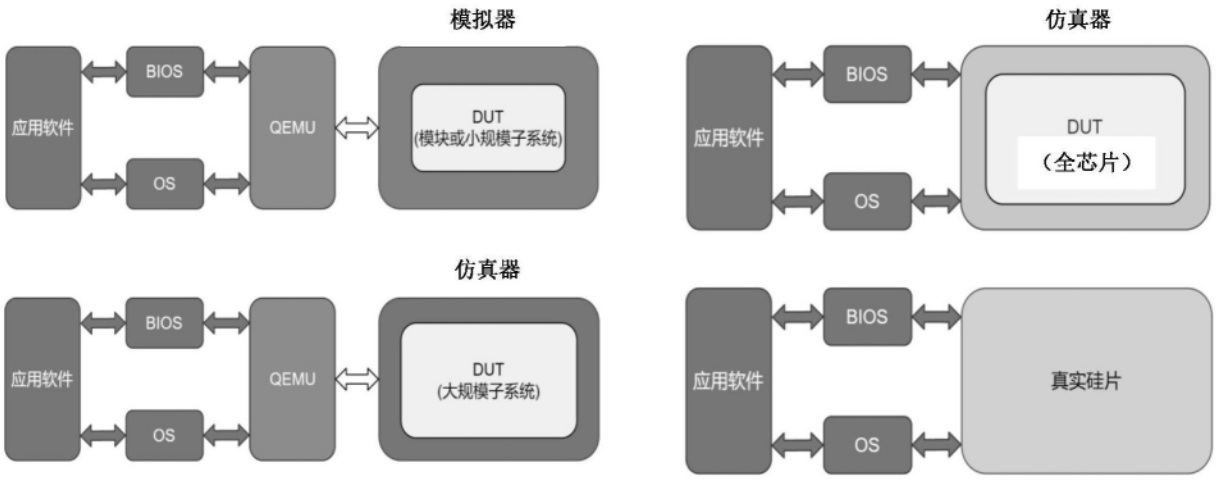

模拟器
应用软件
BIOS
OS
QEMU
DUT
(模块或小规模子系统)
仿真器
应用软件
BIOS
OS
DUT
（全芯片）
仿真器
应用软件
BIOS
OS
QEMU
DUT
(大规模子系统)
应用软件
BIOS
OS
真实硅片

图8