文章编号:1671-4598(2023)10-0255-08 DOI:10.16526/j.cnki.11—4762/tp.2023.10.039 中图分类号:TP391 文献标识码:A

基于模拟串行端口的虚拟设备通信技术研究

侯佩儒，曹炳尧，宋英雄
(上海大学 特种光纤与光接入网重点实验室，上海 200444)

摘要：MBSE是复杂系统设计的重要范式，尤其对于大型嵌入式系统设计具有重大意义；但在MBSE的虚拟验证环节，针对节点的串行端口的互联通信仍缺乏有效灵活的手段；为解决该问题，提出了一种基于Linux系统内核驱动的串行端口模拟方式，实现了基于模拟串行端口的多虚拟设备间的通信技术；分析了标准串行端口的内核驱动架构，进行规范的驱动设计，构建模拟串行端口，以模拟设备替代真实硬件设备；实验验证了模拟的串行端口的功能可用性，且具备平均456.98 Mbps的最大传输速率，满足物理串行端口的速率范围要求；利用该模拟串行端口，实现了虚拟设备间的双向串行通信实验，能够支持嵌入式子系统间串行通信模拟的需要。

关键词：MBSE；复杂嵌入式系统；虚拟设备通信；模拟串行端口；Linux内核驱动

Research on Virtual Device Communication Technology Based on Emulated Serial Port

HOU Peiru, CAO Bingyao, SONG Yingxiong
(Key Laboratory of Specialty Fiber Optics and Optical Access Networks, Shanghai University, Shanghai 200444, China)

Abstract: MBSE is an important paradigm for complex system design, especially for large-scale embedded system design. However, in the virtual validation phase of MBSE, there is still a lack of effective and flexible means for interconnecting communication among nodes' serial ports. To solve this problem, a serial port emulation method based on the Linux system kernel driver is proposed, and a communication technology based on emulated serial ports between multiple virtual devices is implemented. By analyzing the kernel driver architecture of standard serial ports and conducting standardized driver design, emulated serial ports are constructed to replace real hardware devices with emulated devices. The experimental results confirm the functional availability of the emulated serial port, with a maximum transmission rate of 456.98 Mbps on average, meeting the rate range requirements of physical serial ports. Using this emulated serial port, bidirectional serial communication experiments between virtual devices are performed, which can support the needs of emulating serial communication among embedded subsystems.

Keywords: MBSE; complex embedded system; virtual device communication; emulated serial port; Linux kernel driver

0 引言

随着现有的工程系统越发规模大型化、需求复杂化、模块细分化[1]，如：自动驾驶汽车的车载嵌入式系统平均具有1亿行代码，并分布在近100个嵌入式计算机中[2]。传统的设计模式使用文档，难以进行有效的模块划分与跨领域沟通，从而导致研制周期存在较高风险。因此，出现了基于模型的系统工程（MBSE，model-based systems engineering)。其采用标准化建模的方式来支持系统的完整设计，改变了复杂系统的开发模式，尤其是对智慧物联网技术的高速发展催生出的复杂嵌入式系统开发具备重要主导意义。目前，MBSE已广泛应用在航空航天[3-5]、国防军工[6-7]、智慧城市[8]、轨道交通[9]等领域。

基于MBSE的复杂嵌入式系统设计包括需求分析、系统设计、测试验证和需求确认等多个环节[10-11]。其中，虚拟验证在测试验证中尤为重要，但却一定程度上缺乏研究和探索[12]。由于嵌入式系统对硬件的依赖性强、场景专业性高、系统复杂性大，对嵌入式软件开发进行虚拟验证需要在实物集成前提前模拟异构硬件系统，并考虑复杂嵌入式系统中各子系统之间的关联。但是，目前仍缺乏高效通用、考虑整体的嵌入式模拟验证工具或平台。

虚拟化技术的快速发展与广泛应用，解决了单个嵌入式开发板的模拟问题。在本文中，每个利用虚拟化方式模拟的异构子系统称为虚拟设备。虚拟设备是指在MBSE模拟系统中，对应嵌入式系统中可独立分离的物理单元，用于验证开发的软件模拟组件。例如，在汽车部分自动化嵌入式系统中，有Arduino和树莓派两个主从开发板模块，前者连接控制电机的驱动器，后者连接多种传感器进行数据处理与计算。这两个模块使用串行端口进行通信[13-14]，是互为可分离的物理单元，使用虚拟化技术模拟的这两块

收稿日期:2023-04-28； **修回日期**:2023-05-11。
作者简介:侯佩儒(1998-)，女，硕士生。
通讯作者:曹炳尧(1985-)，男，博士，实验师。
引用格式:侯佩儒，曹炳尧，宋英雄. 基于模拟串行端口的虚拟设备通信技术研究[J]. 计算机测量与控制，2023，31(10)：255-262.

模块单元即为虚拟设备。常用的虚拟化技术有操作系统层面虚拟化的 Docker[15]、硬件开发板层面虚拟化的 QEMU (Quick Emulator)[16-17] 等。QEMU 具有跨平台、高速度、可移植等优点，因此，本文选取 QEMU 来创建虚拟设备。

嵌入式系统中，各组件间最普遍的通信方式是基于网络和串行端口的通信。关于虚拟设备间的通信调试，已有一些相关研究。例如，文献［18］利用将 QEMU 中虚拟网卡与宿主机的 tap 虚拟网卡绑定的方式进行虚拟设备网络通信；文献［19］利用源定位技术、基于频率发包与跟踪技术等方法开发了可模拟 CAN 总线的网络仿真开发平台，可进行物理或虚拟控制设备的总线通信测试等。然而，关于串行端口通信调试的研究相对较少，主要是采用物理和虚拟两种方式。物理方式是使用计算机上的真实物理串行端口进行测试，需要使用物理连接线完成端口间的连通，并将真实串行端口配置绑定到虚拟设备内使用。这种方式测试简单、可靠，但是仅支持小规模验证，在大规模场景下时会面临物理接线复杂、计算机的端口资源受限等问题。虚拟方式是采用模拟手段，在宿主机内产生模拟串行端口供虚拟设备测试使用。相较于物理方式，该方式的优点在于不依赖硬件、验证成本低。在 Windows 系统中，已有成熟可用的虚拟串行端口工具 VSPD (virtual serial port driver)。在 Linux 系统中，虽然没有类似的工具，但可以借助 pty (pseudo terminal) 伪终端设备来实现虚拟设备间的通信。但是，不论是成熟工具 VSPD 还是系统自带 pty 设备，都进行了严密的封装，难以灵活接入统计或埋点接口，自定义调试难度较高，导致虚拟设备间的串行端口通信验证较为困难。由上述论述可知，针对虚拟设备互联问题，需要研究灵活可用的串行通信模拟技术，以便实现多组件间的通信调试。

本文针对在 MBSE 虚拟验证环节，嵌入式虚拟设备间的串行端口通信问题展开研究，以实现在 Linux 系统环境下完成虚拟设备间串行通信的目标，并支持复杂嵌入式系统的全面数字化模拟，对 MBSE 工程化的最后一步实施与落地、和我国“十四五”规划深入推进数字化发展都具有重大意义。

1 基于串行端口的虚拟设备间通信方案设计

1.1 需求分析

为了解决复杂嵌入式系统中子系统之间的串行端口通信模拟验证问题，需要首先构建虚拟设备，并创建串行端口，再进行多个虚拟设备之间的通信。如图 1 所示，通信时主要涉及到两类主体对象：虚拟设备、串行端口链路，多个虚拟设备通过串行端口链路实现串行数据传输。

虚拟设备的主要作用是为嵌入式软件提供接近硬件的运行环境，如：具备模拟的嵌入式外围设备、能够利用二进制翻译等技术直接执行异构指令集代码等。由于虚拟设备采用软件逻辑构建模拟，在验证各单元的通信过程中，缺少物理串口介质的链接。而在其中执行串行通信时，必须依赖串行端口，才能保证模拟系统中的程序可以和物理

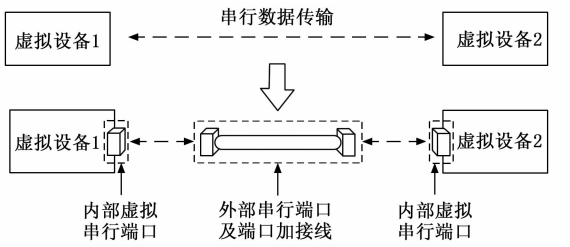

图 1 基于串行端口的虚拟设备间通信方式

设备无缝移植。那就必须要求虚拟设备中具有串行端口，才能进一步对串行端口设备进行开启关闭、发送数据、接收数据等操作。

串行端口链路位于多个虚拟设备之外，由多个外部串行端口及之间的端口连接线构成。其主要作用是接收虚拟设备中内部虚拟串行端口发送的数据，并将数据传输到另一端，发送给另一个虚拟设备，从而为两端虚拟设备的内部虚拟串行端口之间建立可通信的串行链路通道，实现虚拟设备间的串行端口通信。为了使内部虚拟串行端口使用该链路，还需要将内外串行端口进行绑定或映射。

为实现总体方案设计，根据虚拟设备间进行串行端口通信的方式，提出了如图 2 所示的功能需求。

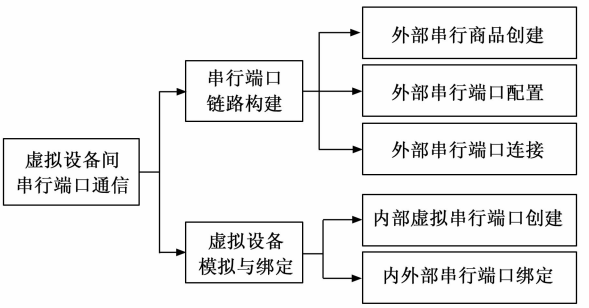

图 2 基于串行端口的虚拟设备间通信功能需求

1.2 总体方案

根据上述功能需求分析，本文提出基于模拟串行端口的虚拟设备间通信架构，如图 3 所示。自下而上分为物理硬件层、虚拟设备层、串行通信层。

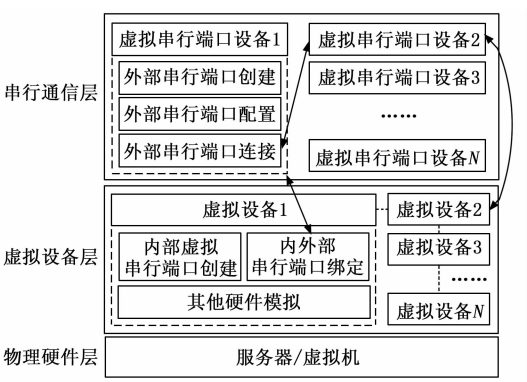

图 3 整体方案架构

物理硬件层：为 Linux 系统的服务器或虚拟机，为上层实现提供必要的整体硬件环境；

虚拟设备层：为多个虚拟设备的模拟，包括内部虚拟串行端口的创建与绑定，以及其他硬件模拟；

串行通信层：为最上层，负责创建、配置、连接外部串行端口，连接后外部串行端口对之间可以构建出串行通信链路，通过内外部串行端口绑定和该串行通信链路，多个虚拟设备间即可实现串行端口通信。

总体方案设计与实现时，主要考虑以下 4 个功能模块的设计：

1）外部串行端口创建模块：

该模块负责在虚拟设备外部创建具备串行端口相关操作功能的设备，以供虚拟设备使用。鉴于已有虚拟串行端口工具具有强封装的特点，难以在其中接入调试、监控、埋点等功能，无法自定义配置，只能进行基本验证。

本方案将自行编码实现开放、可控的虚拟串行端口模拟。为了具备兼容性、可扩展性，以及支持测试代码在无额外修改的情况下的可移植性，采用底层驱动开发的方式，分析 Linux 系统中的标准串行端口驱动，在此基础上类比开发虚拟的串行端口驱动。该种方案下，用户在应用层看到的虚拟串行端口与标准串行端口是一致的，操作也相同，区别仅在于执行底层操作时前者使用模拟方式，而后者使用物理方式。

2）外部串行端口配置：

为模拟创建的外部串行端口配置链路等相关参数。例如：配置该端口需要建立连接的另一个串行端口的索引信息，以便于后续链路构建。

3）外部串行端口连接：

对于多个配置了对端信息的虚拟串行端口，连接对端的作用主要在于构建接收、发送数据的通信链路。可以利用多种方式传递两端发送的消息，例如：管道、共享内存、文件、消息队列、网络等。以文件交互方式为例，每个串行端口维护自己的属性文件，并设置文件的写入回调函数。在发送时将信息写入对端串行端口对应的文件中，对端文件触发写入回调，通知对端端口读取文件，从而实现完整的收发过程。

相较于文件方式，网络传递信息可开发度更高，也更具有分布式扩展性，更适合大规模 MBSE 系统虚拟验证环节的云服务化。因此，本方案采用网络方式实现多个串行端口的连接与链路构建。

4）内部虚拟串行端口创建与绑定模块：

该模块借助成熟的虚拟化软件 QEMU 来实现虚拟设备的整体硬件模拟。并将模拟时创建的内部虚拟串行端口与虚拟设备外、宿主机内的外部串行端口进行绑定，将后者作为前者的设备后端，即可在虚拟设备内使用外部串行端口实现具体功能。

根据上述设计可知，内外部串行端口绑定之后，在虚拟设备内对内部串行端口执行发送、接收等操作都是依靠外部串行端口及其通信链路来完成的。因此，本文接下来的研究重点是基于 Linux 系统底层驱动开发的虚拟外部串行端口模拟和其链路的构建与实现。

2　Linux 系统中标准串行端口驱动分析

在进行模拟串行端口的底层驱动开发之前，需要先对 Linux 系统中标准串行端口的驱动实现进行分析，以标准驱动为参照类比设计与实现，确保系统对虚拟串行端口的兼容性和后续扩展性、可移植性。

2.1　串行端口驱动总述

串行端口是嵌入式系统中常用的硬件端口，有 RS232、RS422、RS485 等多种电气标准[20]。在 Linux 系统中，串行端口是一种 tty（Teletypes）设备，用于串行地输入、输出数据。其设备驱动位于操作系统的内核空间，提供了计算机硬件与应用程序的接口。在内核空间实现虚拟串行端口的模拟，能真正做到上层软件无感知。

Linux 内核中，使用 cdev 结构体描述字符设备，该结构体是所有字符设备的抽象[21]。串行端口的整体驱动框架如图 4 所示。自右至左，驱动可分为 3 个层次，分别为字符设备层、tty 核心层和串行端口硬件层。字符设备层将 tty 设备作为字符设备，提供系统调用的统一接口，即 cdev 结构体的操作函数集。操作函数会进一步调用 tty 核心层的实现函数。tty 核心层对底层硬件的具体形式进行解耦与抽象，负责将上层操作经线路规程进行格式化协议转换后，调用至下层。串行硬件核心层则真正地实现了对设备的管理和操作，直接与底层硬件交互，完成用户请求。

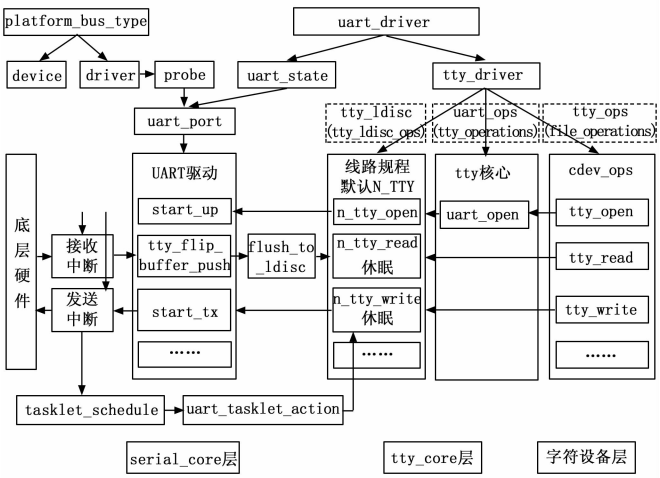

图 4　串行端口驱动框架

自顶层向下，驱动主要包括 uart _ driver、tty _ driver、uart _ state、uart _ port 四大数据结构。uart _ driver 是全局的根数据结构，进行了高度的软件逻辑抽象，负责保存和控制其他所有结构体信息。tty _ driver 是对 tty 层的具体实现；uart _ state 和 uart _ port 是底层驱动的具体实现。整体而言，设备硬件和 uart _ state、uart _ port 一一对应，而一个 uart _ driver 对应一个 tty _ driver 和多个 uart _ state、uart _ port，即多个同类型设备共用一种设备驱动。

完整的串行端口驱动主要包括驱动注册、设备初始化、应用操作 3 个部分。本章以 Linux 系统中 8250 串行端口为

例，依次从这 3 个部分进行分析。

2.2 驱动注册分析

在注册前，8250 串行端口的最上层驱动 uart _ driver 通过 dev _ name 成员指定该设备在 Linux 系统中对应的字符设备文件名称前缀，major 和 minor 成员分别指定设备的主设备号和最大支持设备数量等。

之后，按照标准串行端口的驱动注册流程注册。通过 uart _ register _ driver（）函数向内核注册上述 uart _ driver 定义，并进行一系列初始化操作。为每个 uart _ state 申请空间，初始化 state 的 tty _ port 成员，并为驱动分配与初始化 tty _ driver，设置其操作函数集。该函数集即为系统调用的接口，构建起 Linux 内核驱动和用户空间交互的桥梁。最后进行 tty _ driver 的注册，将其挂载到全局 tty 驱动链表，并进行 proc 文件相关的注册。此时，虽然在内核上注册了驱动，但还没有对接真正的硬件端口。

2.3 设备初始化分析

当硬件设备接入系统时，系统会根据图 4 左上角所示的平台驱动，调用设备独有的 dw8250 _ probe（）硬件初始化函数。该部分与硬件端口紧密相关，但也遵守标准串行端口的设备初始化流程。

8250 串行端口在 uart _ port 之上，封装了具体类型 uart _ 8250 _ port。在初始化时，首先根据调用的硬件平台设备，对其中 uart _ port 结构体成员进行相关初始化。设置中断处理函数、线路规程设置函数、设备类型、寄存器地址等，并分配私有数据等。其次，按照标准串行端口设备初始化流程进一步初始化，主要通过 uart _ add _ one _ port（）函数实现。该函数根据串行端口编号将对应 uart _ state 和该 uart _ port 进行双向绑定，并将设备对应的 uart _ port 添加到 uart _ driver。由此，设备利用硬件初始化函数，通过串行端口硬件层，与 tty 核心层和字符设备层建立了联系，用户空间可以对真实的硬件设备进行操作。

2.4 串行端口应用操作分析

在进行串行端口读写操作之前，需要先打开串行端口。打开流程如图 4 所示，自右至左为系统调用后，内核由字符设备层的 tty _ open（）函数开始逐层调用。直到串行端口硬件层，根据 uart _ port 的操作集调用 8250 串行端口独有的 serial8250 _ startup（）函数，进行串行端口的硬件设置，如波特率、请求发送信号和硬件寄存器设置等，并初始化与使能串行端口的中断。

串行端口打开之后，以读取数据为例分析，主要涉及到两个线程，分别称为前台线程和后台线程。应用程序在用户空间使用 read（）函数读取串行端口数据，经系统调用进入内核空间时由前台线程执行并进行等待。后台线程负责在中断有数据时进行读取，把读取的数据填充至 tty _ buffer 中，再调用 flush _ to _ ldisc（）函数，将数据存放进线路规程的数据接收缓存中，唤醒前台线程，使前台线程读取缓存中的数据，并将数据从内核空间拷贝进用户空间中，即可完成接收。向串行端口写入数据也是类似的，但数据流相反，在此不做赘述。

3 模拟串行端口的方案设计与实现

根据对标准串行端口的驱动分析，能够知道 Linux 系统为各种不同类型的串行端口提供了可复用的 uart 框架。例如，通过封装的根数据结构 uart _ driver，不同的串行端口可以定义自己的驱动信息。如图 5 所示，自定义串行端口驱动在复用已有框架的基础上，需要修改和定义 3 个主要部分。第一部分是每个串行端口驱动注册时都需要定义的，包含设备驱动的名称、设备号等信息；第二和第三部分涉及底层硬件的特定实现，不同的串行端口需提供不同的操作函数以与硬件进行交互。这三部分恰好对应于驱动注册、设备初始化和功能操作。

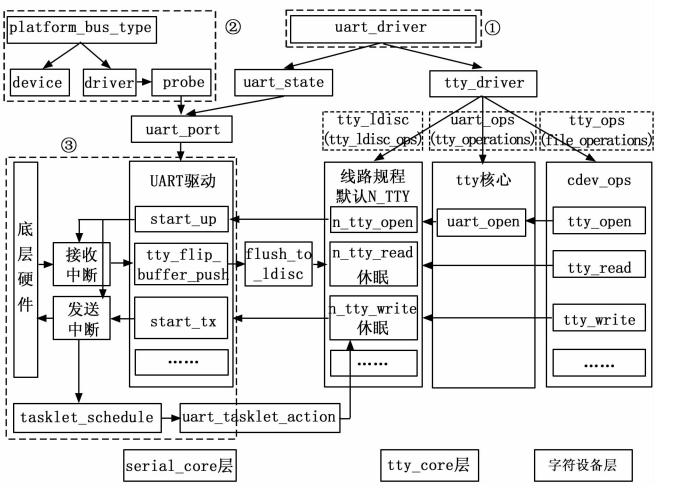

图 5 自定义串行端口驱动设计框架

基于标准串行端口的驱动框架，本章参照 8250 串行端口的定义和注册方式，对总体方案中的外部虚拟串行端口创建、配置、连接模块进行设计，并最终实现了一个完整可用的外部虚拟串行端口驱动。

3.1 虚拟串行端口创建

创建虚拟串行端口需要进行驱动注册和设备初始化。驱动注册确保虚拟设备能在 Linux 系统中被识别，并提供内核空间的接口供用户空间使用；设备初始化类比设备接入的动作，在系统上产生虚拟设备，并进行一系列初始化操作。

1）驱动注册：

参考 8250 串行端口，虚拟串行端口首先需要定义自身的驱动信息。设置驱动名为“virtual-uart”、字符设备文件名称前缀为“vttyU”、最大支持串行端口个数等。并且，在未显式地指定设备号的情况下，Linux 系统会在驱动加载时为其动态分配设备号，以避免人为指定发生冲突。驱动信息定义完成后，通过 uart _ register _ driver（）函数将其注册进内核，并对 uart _ driver 等数据结构进行初始化。后续过程与标准串行端口驱动注册流程相同，进行复用。

2）设备初始化：

在设备创建与初始化之前，需要先定义虚拟串行端口整体的端口数据结构。类比 8250 串行端口在 uart _ port 结

构体之上，进一步封装了与硬件相关的具体类型。在驱动设计时，也为虚拟串行端口封装了具体数据结构 virtual _ uart _ port，包含了 uart _ port。由于虚拟串行端口不具有硬件设备，所以无法提供物理中断和寄存器。为了真实模拟串行端口接收与发送数据的流程，该数据结构内提供接收和发送使能标志位，用于控制虚拟串行端口能否收发，并使用自旋锁保护这两个使能信号。此外，该数据结构还提供了一个工作队列，用于模拟中断处理函数，完成数据中断发送操作。

在具备整体的端口数据结构后，就可以进行具体设备的定义与注册。由于不存在真实设备，无法通过系统自动识别与扫描设备的方式获取设备。因此，需要自行定义多个平台设备，并为每个设备指定不重复的串行端口索引，以便后续串行端口连接时互相识别。定义好设备后，利用 platform _ device _ register（）函数将其注册进内核中。但此时设备并不能进行初始化操作，因为内核中没有该设备对应的平台驱动。

为了保证设备进一步初始化并与整体的端口数据结构建立联系，还需要定义对应的平台驱动。其结构体为 platform _ driver，具有 probe、remove 接口。其中，probe 接口完成设备对应 uart _ port 的注册，remove 接口完成 uart _ port 的注销。并且，需要为平台驱动设置与设备相同的名称来实现设备和驱动间的匹配检测。此外，还需要实现设备注册函数，该函数为 uart _ port 申请内存，设置端口类型、序号、具体的操作函数集等，以及初始化整体数据结构中的自旋锁和工作队列，并指定模拟串行端口发送中断功能的函数作为该工作队列的回调函数。其中，操作函数集和模拟发送中断功能的回调函数是后续实现虚拟串行端口连接的重点内容，将在后续章节具体设计。最后调用 uart _ add _ one _ port（）函数将初始化后的 uart _ port 添加到第一步注册的驱动中。

3.2　虚拟串行端口配置

配置模块主要负责向串行端口配置构建链路相关信息，例如：与该串行端口相连接的另一个串行端口的设备索引，以便串行端口可以自由组合与连接。这也为驱动模块加载进内核后，留下可交互的方式。

在用户空间，可以使用 ioctl（input/output control）函数对设备进行一些特殊控制，实现用户空间和内核驱动之间的沟通。该函数指定设备的文件描述符、请求号和具体参数，其中请求号代表交互协议，设备驱动会根据请求号执行相应操作。用户空间的 ioctl 请求经系统调用后，会调用字符设备层的 tty _ ioctl（）函数，函数内部将进一步根据请求号调用相应控制函数。因此，需要为配置事件预设未被占用、唯一的请求号。

进入内核空间后，在设备初始化时，可以通过 uart _ port 指定的操作函数集实现对应 ioctl（）函数。该函数根据预设的请求号执行期望的配置操作，例如：配置构建链路所需对端信息。首先，获取携带的请求参数，即得到对端串行端口的索引，并将其保存在本设备端口信息中。之后在串行端口连接时，就可以知道连接的对象。如果需要更换连接对象，只需要重新配置即可，使驱动更加灵活。

3.3　虚拟串行端口连接

真实的物理串行端口对是通过外部物理连接线来连通的，虚拟串行端口则只能使用软件的方式进行实现。经过总体方案中对文件、共享内存、管道等实现手段的讨论，为了具有更高可开发度和扩展性，决定采用网络的连接方式。具体实现方式是在设备驱动中设置 uart _ port 的操作函数集，使用网络收发方式完成虚拟串行端口的发送和接收。

使用网络来模拟虚拟串行端口间的通道连接有两种方案，如图 6 所示。

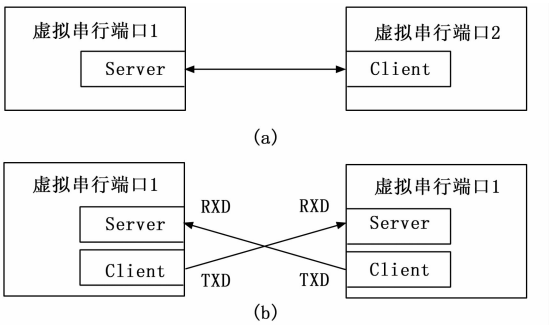

图 6　网络模拟串行端口间通信的方案示意图

第一种方案需要在每个串行端口的整体数据结构中维护一个网络 Socket 套接字成员，并且两个要连接的串行端口要求一个具备服务端，另一个具备客户端，根据配置信息来设置本设备的 Socket 句柄是服务端或是客户端。在串行端口设备初始化时，通过对端索引信息获取对端地址，连接对端 Socket。连接后，将对端 Socket 句柄一并放入整体数据结构中维护，以便于收发数据时直接使用。通过这种方式，可以模拟出串行端口之间的收发传输通道。

第二种方案需要在每个串行端口的整体数据结构中维护两个网络 Socket 套接字成员，其中一个为服务端，用于模拟串行端口的接收数据引脚，另一个为客户端，用于模拟串行端口的发送数据引脚。在这种方式下，两个要连接的串行端口只需要将本端的客户端连接到对端的服务端即可完成连接，并由客户端发送数据，服务端接收数据。通过这种方式，可以模拟出真实的串行端口物理传输方式，即两个串行端口对之间建立的两条网络通道对应了真实串行端口的发送与接收数据两条物理线。

相较于第一种方案，第二种方案更为细化，每个串行端口都是相同且独立的，串行端口之间耦合性更低，无需像第一种方案根据端口索引值设置本端与对端 Socket 套接字是客户端还是服务端，再进行成对绑定。因此，本文选择第二种方案来实现连接模块。

在设备初始化时，根据设备索引值配置监听端口号并开启服务端，等待客户端连接请求。同时，开启一个线程循环判断服务端是否接收到消息。若有消息，则模拟接收中断，将收取到的数据发送到 tty _ buffer 的接收缓存中，并调用 tty

_ flip _ buffer _ push（）函数，将接收缓存中的数据通过线路规程进行接收。在发送数据时，使用设备初始化时配置给工作队列的回调函数来模拟发送中断。在回调函数中，客户端根据对端索引值获取端口号，连接对端串行端口的服务端，并利用网络连接发送数据。即可在没有实际硬件的情况下，使用虚拟串行端口完成数据收发的功能。

4 实验结果与分析

4.1 实验系统

本文使用 12 核 CPU、32GB 内存、200GB 硬盘的虚拟机作为宿主机进行实验，该虚拟机位于搭载 Intel（R）Xeon（R）Silver 4214R CPU @ 2.40GHz 处理器、型号为 UniServer R4900 G3 的服务器上，并使用 QEMU 模拟器来模拟虚拟设备，完成整体方案验证，具体实验系统环境如表 1 所示。

表 1 实验系统环境参数

名称	版本
Ubuntu 操作系统	Ubuntu 18.04.6 LTS
Linux 内核	Linux 5.4.0-144-generic
GCC 编译器	GCC 7.5.0
QEMU 模拟器	QEMU emulator 7.1.0

为了验证方案设计与实现的有效性和正确性，分别对软件模拟的串行端口进行功能和性能验证、对基于模拟串行端口的虚拟设备互联通信进行可行性验证。

4.2 模拟串行端口功能和性能验证

在本方案中，虚拟串行端口驱动被分为设备注册和串行端口整体驱动两部分。前者负责定义硬件设备并将其注册进内核中，以模拟硬件接入；后者负责实现串行端口相关结构的初始化、注册、端口连接等功能，是串行端口的整体设备驱动。这两个驱动模块均具备头文件、模块加载与卸载函数、模块许可声明等规范结构。同时，为驱动模块编写编译文件，内容包括读取内核源码中的编译文件和指明模块源码中各文件的依赖关系等。

编译执行后，会生成对应 .ko 模块文件，可以使用 insmod 命令加载模块。以两个虚拟串行端口为例，驱动模块被加载到系统内核后，将执行初始化程序，开辟内存、新建线程等。此时，如表 2 所示，在系统的/dev 目录下，将会出现两个串行端口字符设备，分别为 vttyU0 和 vttyU1，设备前缀由驱动定义的 dev _ name 指定。

用户态通过执行 ioctl（）函数来指定每个串行端口的对端索引信息。内核态驱动完成相应的设置后，本设备的 Socket 客户端就会连接至对端设备的 Socket 服务端，实现串行端口之间的链路连通。在本实验中，即连接了 vttyU0 和 vttyU1。对于用户态的应用程序而言，这两个字符设备与其他普通串行端口设备并无区别。此时，两个串行端口的连接情况如图 7 所示。编写串行端口读写程序，对其进行功能和性能测试。

1）功能测试。将 vttyU1 作为发送端，vttyU0 作为接收端，进行数据持续收发操作，并将数据内容及传输次数打印到终端。交换发送端与接收端后，结果仍然相同，结果如下所示：

表 2 驱动加载前后系统/dev 目录变化

（a）驱动加载前的/dev 目录

ttyS8	vcsa	vhci
ttyS9	vcsa1	vhost-net
ttyUSB0	vcsa2	vhost-vsock
ttyUSB1	vcsa3	vmci
udmabuf	vcsa4	vsock
uhid	vcsa5	zero
uinput	vcsa6	zfs
urandom	vcsu	
userio	vcsu1	
vcs	vcsu2	

（b）驱动加载后的/dev 目录

ttyS8	vcsa	vhci
ttyS9	vcsa1	vhost-net
ttyUSB0	vcsa2	vhost-vsock
ttyUSB1	vcsa3	vmci
udmabuf	vcsa4	vsock
uhid	vcsa5	vttyU0
uinput	vcsa6	vttyU1
urandom	vcsu	zero
userio	vcsu1	zfs
vcs	vcsu2	

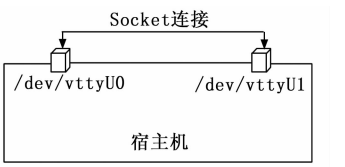

图 7 串行端口测试连接图

（1）vttyU1 发送数据

```
action@action-virtual-machine:~/uart-test sudo ./virtual_uart_test --dev=/dev/vttyU1 --type=write
Send Data: SEND TEST! Num = 1!
Send Data: SEND TEST! Num = 2!
Send Data: SEND TEST! Num = 3!
Send Data: SEND TEST! Num = 4!
Send Data: SEND TEST! Num = 5!
Send Data: SEND TEST! Num = 6!
Send Data: SEND TEST! Num = 7!
Send Data: SEND TEST! Num = 8!
Send Data: SEND TEST! Num = 9!
Send Data: SEND TEST! Num = 10!
```

（2）vttyU0 接收数据

```
action@action-virtual-machine:~/uart-test sudo ./virtual_uart_test --dev=/dev/vttyU0 --type=read
Receive Data: SEND TEST! Num = 1!
Receive Data: SEND TEST! Num = 2!
Receive Data: SEND TEST! Num = 3!
Receive Data: SEND TEST! Num = 4!
```

```
Receive Data: SEND TEST! Num = 5!
Receive Data: SEND TEST! Num = 6!
Receive Data: SEND TEST! Num = 7!
Receive Data: SEND TEST! Num = 8!
Receive Data: SEND TEST! Num = 9!
Receive Data: SEND TEST! Num = 10!
```

实验结果证明，串行端口模拟产生后，在用户态的操作与物理串行端口的操作一致，接收到的数据和对端发送的数据内容相同，因此通信功能可用。

2）性能测试。对串行端口的最大传输速率进行测试，测试方式为：将 vttyU1 作为发送端，以最小的时间间隔，持续发送 1 024 字节的数据给 vttyU0，使得两个串行端口驱动一直处于最大速率的发送与接收状态。在接收端 vttyU0 设置流量计数模块和时间计算模块，得到接收一定数据量时所花费的时间，以计算单位时间内传输的比特数，即数据传输速率。本实验进行 20 组测试，每组测试发送和接收 100 MB 数据，结果如图 8 所示。

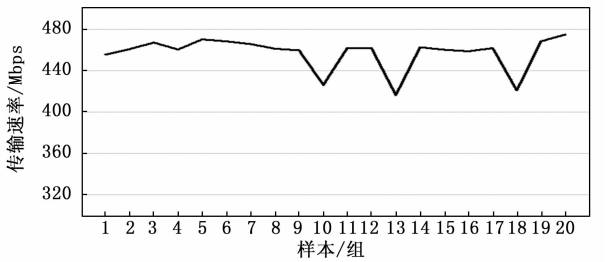

图 8　串行端口传输速率测试结果

实验结果表明，本方案设计的虚拟串行端口驱动的数据传输速率快且较为稳定，最大传输速率介于 416.26 ～ 474.74 mbps 之间，平均值为 456.98 mbps，远超过物理串行端口的传输速率。例如，RS232 串行端口最大传输速率为 20 kbps，RS422/RS485 串行端口最大传输速率为 10 mbps 等。此外，串行端口也在不断发展更高速的传输速率。该虚拟串行端口不仅能够满足基础物理串行端口模拟的速率要求，也能够应对速率更高的增强型串行端口模拟的场景，符合设计需求。

4.3　基于模拟串行端口的虚拟设备间通信验证

对虚拟设备间互联通信进行验证，首先需要创建两个虚拟设备，使用 QEMU 在宿主机内模拟两块树莓派 3B 开发板，包括 CPU、内存、外围设备等硬件资源模拟。其次，每个虚拟设备内都具有串行端口，将前述实验中创建的两个模拟串行端口分别绑定为虚拟设备内串行端口的设备后端，使得两个虚拟设备可以借助外部宿主机内的虚拟串行端口对进行通信。模拟与绑定的具体命令如下所示：

```
    指定模拟树莓派 3B 1024 MB 内存
sudo qemu-system-aarch64 -M raspi3b -m 1024 \
    指定设备树文件
-dtb img/bcm2710-rpi-3-b.dtb
    指定内核
-kernel img/kernel8.img
    指定加载镜像与格式
-drive format=raw,file=img/2020-02-13-raspbian-buster.img
    启动附加命令
-append "rw earlycon=pl011,0x3f201000 console=ttyAMA0 loglevel=8 root=/dev/ mmcblk0p2 fsck.repair=yes net.ifnames=0 rootwait memtest=1 dwc_otg.fiq_fsm_enable=0 8250.nr_uarts=1"
    指定标准输入输出
-serial stdio
    为树莓派启用 USB 键盘鼠标等模拟
-usb -device usb-kbd -device usb-tablet
    绑定外部串行端口/dev/vttyU0(另一个绑定/dev/vttyU1)
-serial /dev/vttyU0
```

绑定完成后，启动虚拟设备，进入系统后可以看到在/dev 目录下都多出一个设备 ttyS0。该设备是串行端口的设备前端，将消息在虚拟设备和设备后端之间进行转发，真实地与虚拟设备外进行数据收发交互的仍为宿主机内创建的虚拟串行端口。由此，两块虚拟开发板借助外部虚拟串行端口对 vttyU0 和 vttyU1 建立了两个内部 ttyS0 的通信链路，连接情况如图 9 所示。

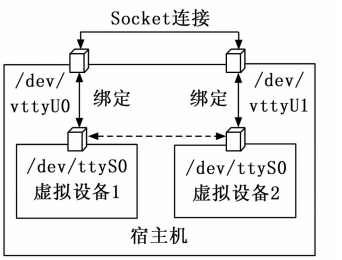

图 9　虚拟设备串行通信测试连接图

对其进行收发测试，测试方式为：在一块虚拟树莓派开发板内部打开串行端口 ttyS0，发送数据并打印发送次数，另一块虚拟树莓派开发板内部打开串行端口 ttyS0，进行数据接收并输出至终端。反过来通信也是相同的，测试结果如下所示：

（1）虚拟设备 1 发送数据

```
pi@ raspberrypi: ~/myTests/uart-test ./virtual-uart --dev=/dev/ttyS0 --type=write
Send Data: SEND TEST! Num = 1!
Send Data: SEND TEST! Num = 2!
Send Data: SEND TEST! Num = 3!
Send Data: SEND TEST! Num = 4!
Send Data: SEND TEST! Num = 5!
Send Data: SEND TEST! Num = 6!
Send Data: SEND TEST! Num = 7!
Send Data: SEND TEST! Num = 8!
Send Data: SEND TEST! Num = 9!
Send Data: SEND TEST! Num = 10!
```

（2）虚拟设备 2 接收数据

```
pi@ raspberrypi: ~/myTests/uart-test ./virtual-uart --dev=/dev/ttyS0 --type=read
Receive Data: SEND TEST! Num = 1!
```

Receive Data: SEND TEST! Num = 2!
Receive Data: SEND TEST! Num = 3!
Receive Data: SEND TEST! Num = 4!
Receive Data: SEND TEST! Num = 5!
Receive Data: SEND TEST! Num = 6!
Receive Data: SEND TEST! Num = 7!
Receive Data: SEND TEST! Num = 8!
Receive Data: SEND TEST! Num = 9!
Receive Data: SEND TEST! Num = 10!

测试结果表明，虚拟设备间可以利用本方案实现的虚拟串行端口，建立传输通道，进行双向通信。该程序也可以直接移植到物理的嵌入式终端上，直接进行串行设备通信，可以确保模拟系统和程序具有较高的保真度和移植性。

5 结束语

本文针对在 MBSE 系统中的虚拟设备间进行串行互联通信的需求，提出了一种针对 Linux 系统的软件模拟串行端口方案，并基于模拟串行端口实现多个嵌入式虚拟设备之间的互联通信。本文的串行端口模拟通过参照 Linux 系统的标准串行端口驱动，复用标准 uart 驱动框架，完成内核驱动开发，从内核态产生虚拟串行端口，并实现了参数配置、通道连接等功能。

通过对两个已连接的虚拟串行端口进行通信测试，验证了本文提出的模拟串行端口的可用性和具备平均 456.98Mbps 的最大传输速率，能够满足物理串行端口模拟的传输速率要求。

同时，通过为 QEMU 构建的虚拟设备绑定模拟串行端口，实现了虚拟设备之间的串行数据通信。基于虚拟设备开发的串行通信程序，可直接移植到物理硬件设备上，提高了嵌入式系统开发调试的效率，为复杂嵌入式系统的全面数字化模拟提供了支持。

参考文献：

[1] GREGORY J, BERTHOUD L, TRYFONAS T, et al. The long and winding road: MBSE adoption for functional avionics of spacecraft [J]. Journal of Systems and Software, 2020, 160: 110453.

[2] SIRGABSOU Y, BARON C, PAHUN L, et al. Model-driven engineering to ensure automotive embedded software safety. Methodological proposal and case study [J]. Computers in Industry, 2022, 138: 103636.

[3] ELAKRAMINE F, JARADAT R, HOSSAIN N U I, et al. Applying systems modeling language in an aviation maintenance system [J]. IEEE Transactions on Engineering Management, 2021, 69 (6): 4006 - 4018.

[4] 高金艳，汪路元，潘忠石，等. 火星维护与管理装置的 MBSE 架构建模 [J]. 系统工程与电子技术，2023，45 (5)：1441 - 1450.

[5] ANYANHUN A I, ANZAGIRA A, EDMONSON W W. Intersatellite communication: An MBSE operational concept for a multiorbit disaggregated system [J]. IEEE Journal on Miniaturization for Air and Space Systems, 2020, 1 (1): 56 - 65.

[6] 崔展博，景 博，焦晓璇，等. MBSE 技术在飞控系统 PHM 中的应用研究 [J]. 国外电子测量技术，2022，41 (3)：69 - 78.

[7] 王国梁，赵 滟，卢志昂，等. 基于 MBSE 的导弹武器系统效能评估 [J]. 火力与指挥控制，2022，47 (8)：116 - 123.

[8] JOSHI N, MITTAL M, JAYAWANT Y, et al. Applying systems engineering framework for architecting a smart parking system within a smart city [J]. INCOSE International Symposium, 2021, 31 (1): 16 - 30.

[9] 何 旭，张海柱，丁国富，等. 基于 MBSE 的高速列车转向架系统设计建模方法 [J]. 机械设计与研究，2022，38 (4)：179 - 185.

[10] RAMOS A L, FERREIRA J V, BARCELó J. Model - based systems engineering: an emerging approach for modern systems [J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2011, 42 (1): 101 - 111.

[11] 姬晓慧，陈国定. MBSE 方法论实施方法研究 [J]. 中国新技术新产品，2022 (4)：33 - 38.

[12] SURYADEVARA J, TIWARI S. Adopting mbse in construction equipment industry: An experience report [C] //2018 25th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2018: 512 - 521.

[13] PASUPULETI V S M R, JOLLU B M, JANAPATI K C, et al. Partial automation of automobiles using embedded systems [C] // 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184), IEEE, 2020: 191 - 195.

[14] KUMAR S, JASUJA A. Air quality monitoring system based on IoT using Raspberry Pi [C] // 2017 International conference on Computing, Communication and Automation (ICCCA), IEEE, 2017: 1341 - 1346.

[15] RAD B B, BHATTI H J, AHMADI M. An introduction to docker and analysis of its performance [J]. International Journal of Computer Science and Network Security (IJCSNS), 2017, 17 (3): 228.

[16] MAMBU K, CHARLES H P, DUMAS J, et al. Instruction set design methodology for in-memory computing through QEMU-based system emulator [C] //2021 IEEE International Workshop on Rapid System Prototyping (RSP), IEEE, 2021: 43 - 49.

[17] JOSHI C V. Software emulation of STM32 controller for virtual embedded design/test environment [D]. Eindhoven University of Technology, 2019.

[18] 李汶洁. 基于 QEMU 的虚拟网络设备物理层通信的研究 [J]. 电子技术与软件工程，2016 (2)：34 - 35.

[19] 昌 路，郭永红，关永峰，等. 一种开放式 CAN 总线网络仿真开发平台的实现 [J]. 系统仿真学报，2014，26 (6)：1236 - 1243.

[20] PWINT H N Y, KYWE T, AUNG T T E. PC and PIC based electronic devices controller using serial communication [J]. International Journal of All Research Writings, 2019, 2 (3): 129 - 133.

[21] 郑 强. Linux 驱动开发入门与实战 [M]. 北京：清华大学出版社，2014.