http://myfj.cnjournals.com myfj_sadri@163.com (021)20866796

DOI: 10.19416/j.cnki.1674-9804.2023.04.020

基于 QEMU 仿真的 GIC 中断处理技术研究

王 宁* 王珍珍 崔西宁

(中国航空工业集团公司西安计算技术研究所,西安 710076)

摘 要: QEMU6.1.0 版本下,基于 ARM 体系架构开发的 VIRT 模拟器,在运行天脉操作系统时无法正确触发 GIC 中断控制器产生中断的问题,分析了 VIRT 模拟器对于 GIC 中断控制器的初始化以及触发操作,同时查阅了 GIC 中断控制器的相关说明文档,进一步分析天脉操作系统中有关中断控制器的初始化代码,找出了 VIRT 模拟器与天脉操作系统在 GIC 中断控制器初始化和使用方式上的具体差异。通过修改 VIRT 模拟器中与 GIC 中断控制器相关的代码完成对天脉操作系统的适应。在不改动天脉操作系统代码的情况下,顺利将 VIRT 模拟器模拟的 GIC 中断控制器驱动运行成功。

关键词: QEMU 仿真;GIC 中断控制器;天脉操作系统;VIRT 模拟器

中图分类号: TP391.9 **文献标志码**: A **OSID**:

0 引言

QEMU(quick emulator)[1]是一款开源的硬件环境模拟仿真软件,它采取动态二进制翻译技术将目标处理器的指令集翻译为等价的本地处理器指令组合,从而实现对硬件环境的异构模拟仿真。全球各地的开源软件开发人员基于技术成熟度高、市场占有率大的硬件产品,为 QEMU 贡献了大量的硬件环境模拟代码。这些代码对于商业产品实现了非常精细的模拟,但是对于专业领域(例如:航空航天领域等)所使用的一些类似硬件环境,并不能做到无缝衔接的程度[4-5]。

天脉操作系统是航空工业计算所研发的,拥有完全自主知识产权的嵌入式实时操作系统产品,可被应用于综合化航电系统、分区隔离操作环境以及单核/多核处理器等多种嵌入式应用场景。天脉嵌入式实时操作系统产品目前已在航空航天各个领域被广泛使用,覆盖了数据采集、通信导航、图形图像以及综合任务处理等诸多应用场景。

将为 ARM 多核处理器开发的嵌入式天脉操作系统运行在 QEMU(版本号:6.1.0)[1]的 VIRT 模拟器上时,发现能够成功运行在真实硬件环境下的操作系统,并不能将 VIRT 模拟器中所模拟的 GIC 中断控制器驱动运行。具体表现为:无法产生任何中断信号。

本文通过深入分析 GIC 中断控制器的说明文档、VIRT 模拟器对 GIC 中断控制器的硬件模拟代码以及天脉操作系统对 GIC 中断控制器的驱动代码,找出了操作系统驱动代码与模拟器硬件模拟代码之间的匹配差异。通过修改模拟器的硬件模拟代码,完成对操作系统驱动代码的适配。最终在不改动天脉操作系统驱动代码的情况下,顺利将 VIRT 模拟器模拟的 GIC 中断控制器成功驱动。

修改后的 VIRT 模拟器很好地适配了目标操作系统。在一些暂时缺少真实硬件的场合,可以将 VIRT 模拟器先提供给上层软件开发用户,提前进行应用软件的开发调试工作,做到了软硬件的同步开发,大大节省了后期软硬件联调联试所需的时间,对于工程项目的快速交付具有极为重要的推动作用[6]。

1 GIC 中断控制器的配置

本文所涉及的 GIC 中断控制器为目前 ARM 架

* 通信作者. E-mail: wwnn1_class@sina.com

引用格式: 王宁,王珍珍,崔西宁. 基于 QEMU 仿真的 GIC 中断处理技术研究[J]. 民用飞机设计与研究,2023(4):141-145. WANG N,WANG Z Z,CUI X N. Research on GIC interrupt processing technology based on QEMU simulation[J]. Civil Aircraft Design and Research,2023(4):141-145(in Chinese).

构中应用最广泛的 V3 版本[2-3],所涉及的配置寄存器类包括:

1) GICC:处理器相关的寄存器类。

2) GICD:中断分发相关的寄存器类。

下面首先分析 GIC 中断控制器说明文档中相关的配置描述。

1.1 GICC 相关的配置寄存器

GICC 寄存器类是中断控制器与 CPU 相关的接口寄存器组(CPU interface registers)。在 ARM 体系架构中,GICC 寄存器类采用系统寄存器方式进行访问,即通过 MRC 指令以及相应的寄存器编码进行访问。同时,GICC 部分寄存器也可以通过内存映射的方式进行访问,即寄存器有自己的内存地址,CPU 采用内存读写的方式对寄存器进行访问。

GICC 寄存器组中与本文有关的寄存器如表 1 所示,ICC_* 表示采用系统寄存器方式进行访问。

表 1 GICC 寄存器(部分)列表

名 称	类 型	描 述
ICC_BPR0	可读可写	Group0 的优先级分段寄存器
ICC_BPR1	可读可写	Group1 的优先级分段寄存器
ICC_CTLR	可读可写	CPU 接口控制寄存器
ICC_IGRPEN0	可读可写	Group0 中断使能寄存器
ICC_IGRPEN1	可读可写	Group1 中断使能寄存器
ICC_PMR	可读可写	中断优先级屏蔽寄存器
ICC_SRE	可读可写	系统寄存器访问控制寄存器
ICC_IAR0	只 读	Group0 分组产生的中断号
ICC_IAR1	只 读	Group1 分组产生的中断号

CC_BPR0、ICC_BPR1 以及 ICC_PMR 负责优先级相关的配置;ICC_IGRPEN0 与 ICC_IGRPEN1 负责寄存器组 Group0 与 Group1 的使能配置;ICC_SRE 负责 GICC 寄存器的访问方式配置;ICC_CTLR 负责对中断行为及属性进行配置(写)或访问(读);ICC_IAR0 与 ICC_IAR1 负责在产生中断时提供具体的中断号。

1.2 GICD 相关的配置寄存器

GICD 寄存器类是控制中断控制器中断分发功能的寄存器组(distributor registers)。在 ARM 体系架构中,GICD 寄存器类采用内存映射的方式进行访问,即该类中的每个寄存器都有自己的内存地址,CPU 采用内存读写的方式对寄存器进行访问。

GICD 寄存器组中与本文有关的寄存器如表 2 所示。

表 2 GICD 寄存器(部分)列表

名 称	类 型	描 述
GICD_CTLR	可读可写	中断分发控制寄存器
GICD_IGROUPR	可读可写	中断分组寄存器
GICD_ISENABLER	可读可写	中断投递使能寄存器
GICD_ICENABLER	可读可写	中断投递禁止寄存器
GICD_IPRIORITYR	可读可写	中断优先级配置寄存器
GICD_ICFGR	可读可写	中断触发方式配置寄存器
GICD_TYPER	只 读	中断控制器类型寄存器

GICD_CTLR 负责配置中断和路由使能;GICD_IGROUPR 负责分配中断所属的 Group(Group0 或 Group1);GICD_ISENABLER 负责配置中断投递使能;GICD_ICENABLER 负责配置中断投递禁止;GICD_IPRIORITYR 负责中断的优先级配置;GICD_ICFGR 负责中断触发方式的配置(沿触发或者电平触发);GICD_TYPER 负责提供中断控制器的各种属性信息。

2 天脉操作系统对 GIC 中断控制器的配置

2.1 对 GICC 寄存器组的配置

天脉操作系统对 GICC 寄存器组的关键配置包括:

1) 通过配置 ICC_SRE 寄存器,使能系统寄存器访问方式;

2) 通过配置 ICC_PMR 寄存器,设置中断优先级的屏蔽字;

3) 通过配置 ICC_BPR1 寄存器,设置 Group1 中断组的优先级分段标志;

4) 通过配置 ICC_CTLR 寄存器,设置中断访问清除为 EOI 模式;

5) 通过配置 ICC_IGRPEN1 寄存器,使能 Group1 中断组。

通过检查相关代码发现天脉操作系统并没有对 ICC_IGRPEN0 与 ICC_BPR0 进行配置。

2.2 对 GICD 寄存器组的配置

天脉操作系统对 GICD 寄存器组的关键配置包括:

1）通过读取 GICD_TYPER 寄存器,获取 GIC 中断控制器的版本以及最大中断号等信息;

2）通过配置 GICD_ICFGR 寄存器,设置所有中断的触发方式为电平触发;

3）通过配置 GICD_IPRIORITYR 寄存器,设置中断的优先级;

4）通过配置 GICD_ICENABLER 寄存器,将所有中断初始状态设置为禁止;

5）通过配置 GICD_CTLR 寄存器,使能中断路由功能以及 Group1 中断组。

通过检查相关代码,发现天脉操作系统并没有对 GICD_IGROUPR 进行配置。

2.3　模拟和真实硬件配置结果的差异

通过以上描述可知,操作系统在配置 GIC 中断控制器时,只设置了部分寄存器的值,其它寄存器则采用了系统上电后的默认值。因此,极可能是由于真实硬件与 VIRT 模拟器对于 GIC 中断控制器各个寄存器上电缺省值的设置不同,导致了操作系统在两个平台上运行时产生了不同的结果。

为了对比天脉操作系统在两个平台上对 GIC 中断控制器最终的配置结果,在配置完成之后加入了对表 1、表 2 中关键寄存器值的打印。经过对比两个平台的打印输出信息发现:VIRT 模拟器平台在配置完成之后,将所有中断全部归为 Group0 中断组;而硬件平台在配置完成之后,将所有中断全部归为 Group1 中断组[9]。

由 2.1 节可知,在操作系统配置代码中,并没有对 Group0 中断组进行配置(涉及 ICC_IGRPEN0 与 ICC_BPR0 寄存器)。进一步检查操作系统的中断响应查询代码发现,在外部中断产生之后,需要获取具体的中断号时,操作系统也只是读取了 Group1 中断分组的中断标识寄存器(ICC_IAR1),并未读取 Group0 中断分组的中断标识寄存器(ICC_IAR0)。

根据以上分析可知,由于真实硬件和 VIRT 模拟器对于中断的默认分组不同,而操作系统的所有配置访问代码都是基于 Group1 分组进行操作,这就导致在 VIRT 模拟器上运行天脉操作系统时,并不能正确驱动 GIC 中断控制器。

3　VIRT 模拟器对中断控制器的模拟

通过分析 VIRT 模拟器与 GIC 中断控制器相关的初始化代码发现,模拟器在 ARM 处理器的安全模式下将中断归为 Group0 中断组,在非安全模式下将中断归为 Group1 中断组[7]。相关代码如图 1 所示。

```
typedef struct GICState {
    ....
    /* configure IRQs as group 1 (NS) on reset? */
    bool irq_reset_nonsecure;
    ....
} GICState;

....

/* If we're resetting a TZ-aware GIC as if secure firmware
 * had set it up ready to start a kernel in non-secure, we
 * need to set interrupts to group 1 so the kernel can use them.
 * Otherwise they reset to group 0 like the hardware.
 */
if (s->irq_reset_nonsecure) {
    cs->gicr_igroupr0 = 0xffffffff;
} else {
    cs->gicr_igroupr0 = 0;
}

....

if (s->irq_reset_nonsecure) {
    for (i = GIC_INTERNAL; i < s->num_irq; i++) {
        gicv3_gicd_group_set(s, i);
    }
}
```

图 1　VIRT 模拟器对中断分组的代码

但是在 VIRT 模拟器 GIC 设备的创建过程中并没有将控制中断分组的变量 irq_reset_nonsecure 设置为 TRUE,最终导致所有中断被归为 Group0 分组。正确的做法应该是根据当前 ARM 处理器的安全模式,将 irq_reset_nonsecure 设置为正确的值。修改后的 GIC 设备创建代码如图 2 所示。

```
static void create_gic(VirtMachineState *vms){
    GICv3State *gicv3s = ARM_GICV3_COMMON(vms->gic);
    ....

    /* 根据处理器的安全状态设置irq_reset_nonsecure */
    if ( !vms->secure ){
        gicv3s->irq_reset_nonsecure = true;
    }
    else{
        gicv3s->irq_reset_nonsecure = false;
    }

    ....
}
```

图 2　VIRT 模拟器 GIC 设备创建代码

修改上述代码之后,再次在 VIRT 模拟器平台运行天脉操作系统,并打印 GIC 中断控制器最终的配置结果。打印输出信息显示此时的配置结果与真实硬件平台完全一致。说明上述对代码的改动是符合预期的。

虽然经过上述修改,使模拟平台与真实硬件平

台对 GIC 中断控制器的配置完全一致,但是中断信号依然不能成功送达 CPU。进一步分区 VIRT 模拟器代码发现,在模拟中断发送环节,模拟器根据中断类型的不同,对中断号进行了二次调整。

在 ARM 多核处理器架构下,GIC 中断控制器将中断分为共享中断(SPI)和私有中断(PPI)两类。SPI 为所有处理器核共享,PPI 为每个处理器核私有[2-3,8]。

模拟器对不同类型中断号的调整代码如图 3 所示。

```
static void gicv3_set_irq(void *opaque, int irq, int level){
    GICv3State *s = opaque;

    ....
    if (irq < (s->num_irq - GIC_INTERNAL)) {
        /* external interrupt (SPI) */
        gicv3_dist_set_irq(s, irq + GIC_INTERNAL, level);
    } else {
        /* per-cpu interrupt (PPI) */
        int cpu;

        irq -= (s->num_irq - GIC_INTERNAL);
        cpu = irq / GIC_INTERNAL;
        irq %= GIC_INTERNAL;
        assert(cpu < s->num_cpu);
        ....
        gicv3_redist_set_irq(&s->cpu[cpu], irq, level);
    }
}
```

图 3　VIRT 模拟器对中断号的调整代码

因为 PPI 为每个处理器核所私有,所以也称为核的内部中断。本文所涉及的硬件和模拟平台中,ARM 处理器的每个核有 32 个内部中断,中断号都是从 1 开始到 32 结束。SPI 被处理器的所有核所共享,中断号从 33 开始到 288 结束共 256 个。

由图 3 所列的代码可知,VIRT 模拟器在触发 SPI 中断之前,会在传入的中断号基础上,增加内部中断的个数(32),然后以新调整的中断号去触发中断,将其发送给处理器的所有核。而在触发 PPI 中断之前,VIRT 模拟器则会根据传入的中断号,首先算出对应的 CPU 核号,以及真正的 PPI 中断号(1-32 之间),然后把调整后的中断号发送给对应的处理器核。

分析总结 VIRT 模拟器对于中断号的调整,可得出以下结论:

1) VIRT 模拟器中的 SPI 中断号从 1 到 256,对应真实硬件的中断号从 33 到 288;

2) VIRT 模拟器中的 PPI 中断号,从 257 开始,每 32 个对应一个处理器的核,即:257-288 属于核 0;289-320 属于核 1;依次类推。

基于以上结论,在 VIRT 模拟器中设备挂接中断时,不能直接使用真实硬件的中断号[10],而应该采取以下调整策略:

1) 对于 SPI 中断,挂接时应该在真实的物理中断号的基础上减去 32。即:

$$INT_{模拟} = INT_{物理} - 32 \quad (1)$$

2) 对于 PPI 中断,挂接时应该以真实的物理中断号,计算出对应的模拟器中断号后再使用。计算公式为:

$$INT_{模拟} = 256 + 32 \times CoreId + INT_{物理} \quad (2)$$

经过上述的调整之后,在 VIRT 模拟器上运行天脉操作系统时,中断便可以正常产生。

4　结论

天脉操作系统对 GIC 中断控制器的初始化遵守了 GIC 体系架构文档中的描述。在真实硬件平台运行天脉操作系统时,基于 ARM 处理器的非安全状态以及 GIC 中断控制器相关寄存器的默认值,所有中断被划分为 Group1 中断组。而在 VIRT 模拟器中,所有中断默认被划分为 Group0 中断组。另外,在中断投送过程中,VIRT 模拟器对中断号进行了二次调整,导致设备挂接的中断号与最终投送给处理器的中断号并不一致。在调整中断分组以及设备挂接的中断号之后,天脉操作系统即能够成功在 VIRT 模拟器平台运行。但是,如果设备挂接的中断号与实际投送的中断号不一致,则会给用户带来不好的使用体验。如何通过修改 VIRT 模拟器代码使得二者一致,是需要进一步研究的内容。

另外,在实际应用中,操作系统与 VIRT 模拟器会因虚拟设备模型、操作系统底层驱动、设备设计手册相互不一致等问题无法匹配,因此,从工程角度建议操作系统和 VIRT 模拟器的不兼容可从这三个方面同时出发,进行操作系统与 VIRT 模拟器适配工作。

参考文献:

[1] The QEMU Project Developers. QEMU documentation: Release 6.1.0[S/OL]. (2021-08-24)[2023-06-07]. www.qemu.org.

[2] ARM Limited. Arm generic interrupt controller architecture specification: GIC architecture version 3.0 and version 4.0[S/OL]. (2021-02-12)[2023-06-07]. www.arm.com. https://download.csdn.net/download/

adaptiver/10392719? utm_source=bbsseo.

[3] ARM Limited. ARM architecture reference manual: ARMv8, for ARMv8-A architecture profile [S/OL]. (2013-12-24) [2023-06-07]. www.arm.com. https://download.csdn.net/download/qq_30025621/11129198? utm_medium=distribute.pc_relevant_download.none-task-download-2~default~BlogCommendFromBaidu~Rate-2-11129198-download-9992241.257%5Ev14%5Epc_dl_relevant_base1_c&depth_1-utm_source=distribute.pc_relevant_download.none-task-download-2~default~BlogCommendFromBaidu~Rate-2-11129198-download-9992241.257%5Ev14%5Epc_dl_relevant_base1_c&spm=1003.2020.3001.6616.2.

[4] 张磊,陈程,林卓,等. 基于 QEMU 的 AARCH64 平台仿真技术的研究[J]. 电脑编程技巧与维护,2023(4):120-122.

[5] ALMER O, BÖHM I, VON KOCH T E, et al. Scalable multi-core simulation using parallelldynamic binary translation[C]//2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, July 18-21,2011, Samos, Greece. [S.l.]:IEEE, 2011: 190-199.

[6] WANG K C. Multiprocessing in embedded systems[J]. Embedded and Real-Time Operating Systems, 2017: 329-399.

[7] DALL C, NIEH J. KVM/ARM: the design and implementation of the linux ARM hypervisor[J]. ACM SIGPLAN Notices: A monthly publication of the special interest group on programming languages, 2014, 49(4): 333-348.

[8] 林育斌. 基于 QEMU 的 BM3803MG 处理器模拟器的研究与实现[D]. 北京:北京邮电大学,2018.

[9] NING Z Y, ZHANG F W. Hardware-assisted transparent tracing and debugging on ARM[J]. IEEE Transactions on Information Forensics and Security, 2019, 14(6): 1595-1609.

[10] DALL C. The design, implementation, and evaluation of software and architectural support for ARM virtualization [D]. New York: Columbia University, 2018.

作者简介

王　宁　男,本科,高级工程师。主要研究方向:机载计算机软件。E-mail: wwnn1_class@sina.com

王珍珍　女,硕士,助理工程师。主要研究方向:机载计算机软件。E-mail: 1072593676@qq.com

崔西宁　男,工学博士,研究员。主要研究方向:机载计算机软件。E-mail: cxning@avic.com

Research on GIC interrupt processing technology based on QEMU simulation

WANG Ning*　WANG Zhenzhen　CUI Xining

(Xi'an Aeronautics Computing Technique Research Institute, AVIC, Xi'an 710076, China)

Abstract: GIC is Generic Interrupt Controller of ARM architecture. VIRT emulator in QEMU (version 6.1.0) was developed for hardware emulation of ARM architecture. However, the TianMai operating system developed for ARM architecture cannot drive the emulated GIC correctly when running on VIRT emulator, while the exact same operating system can drive GIC correctly running on real hardware environment. By analyzing the emulation source code about GIC in VIRT emulator, the specification document of GIC, and the driver of GIC in TianMai operating system, the differences about manner of GIC operation between TianMai operating system and VIRT emulator were found. To eliminate these differences, source code about GIC emulating in VIRT emulator was modified. Finally, TianMai operating system can drive the emulated GIC correctly when running on newly created VIRT emulator, without making any change in TianMai operating system itself.

Keywords: QEMU emulation; generic interrupt controller; TianMai operating system; VIRT emulator

* Corresponding author. E-mail: wwnn1_class@sina.com