Zephyr RTOS软硬件适配与虚拟化技术

汇报人：谢国琪

嵌入式与网络计算湖南省重点实验室（湖南大学）

2023-08-12

CONTENTS

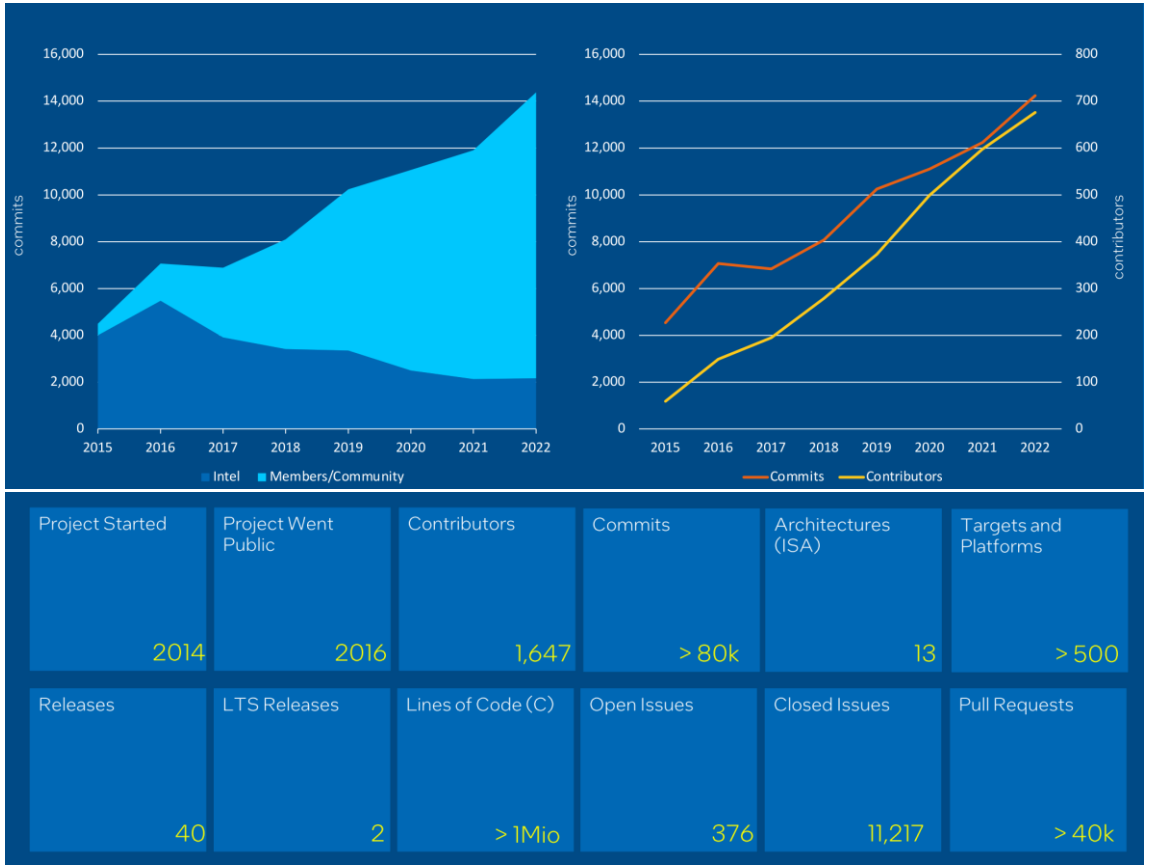
目录

01

Zephyr RTOS基本介绍

Linux基金会下的开源实时操作系统Zephyr

- 活跃的开源社区，**开源贡献者超过1600个、提交数量达到80k+**；
- 支持不同硬件，超过**13种硬件架构**；
- 支持超过**500+的实际板卡**；
- Zephyr是一个产品级应用，发布了**40个release**版本，包含**2个LTS版本**；
- 良好的**实时性和安全性**支持；
- 支持多种嵌入式场景下的**物联网协议。**



Zephyr 开源项目数据（2023）

Zephyr RTOS基本介绍

Zephyr架构特性

➢ 内核

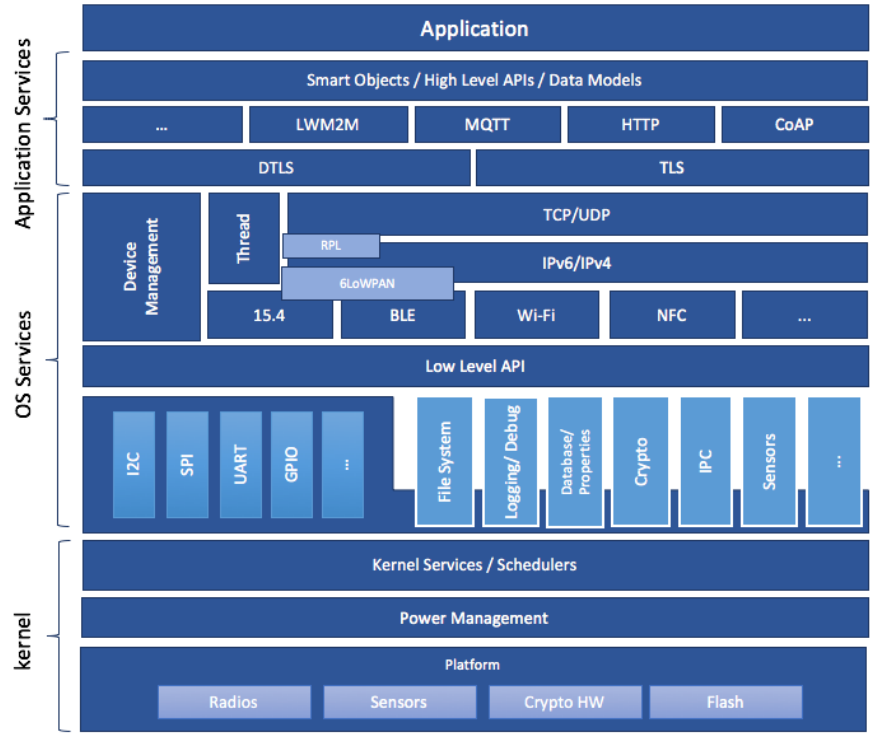
- 提供Kernel service，例如thread，同步，数据传递，中断管理，时间管理，内存管理等；进行任务调度；电源管理；平台相关的特殊驱动，例如Radios，传感器，加密硬件，Flash等。

➢ 系统服务

- 设备驱动实现，提供统一的底层驱动接口；设备管理；网络L2实现及接口抽象；网络层和传输层协议栈(TCP/IP)，socket接口。

➢ 开源性

- 依托于Linux基金会，技术开源，开发者众多，采用Apache 2.0协议许可。Zephyr 有一个充满活力的国际开发社区，它和物联网操作系统中的 ARM Mbed OS、nuttX 和 RIOT 比较，活跃度很高。



Zephyr OS 系统架构

CONTENTS

目录

02

Zephyr适配RK3568

背景

- RK3568是瑞芯微（Rockchip）推出的一款高性能嵌入式处理器芯片。它是RK356x系列的一员，该系列针对物联网设备、智能音箱、工业控制和嵌入式计算等领域的应用提供了解决方案。
- ARMv8.1架构
- ZVM的硬件适配、Paddle Lite的硬件适配工作都需要在RK3568上面实现。

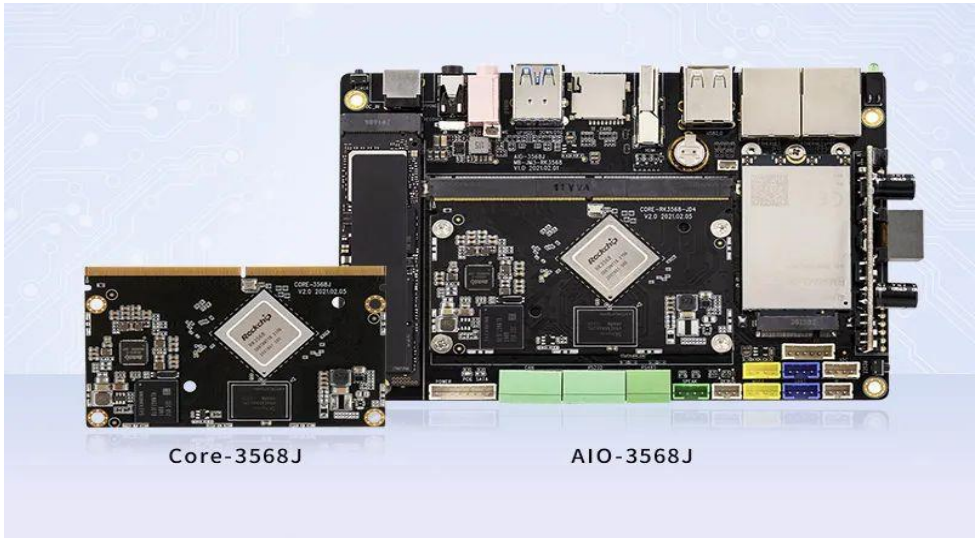
RK3568：人气爆款，炙手火热

嵌入式计算湖南省重点实验室 2023-07-29 22:36 发表于湖南

众所周知，RK3568是目前AIoT领域最受欢迎的芯片之一，成为了众多领域采用的人气爆款，是炙手火热的中高端产品，其出色的性能和多样化的应用领域备受关注。在工艺上，采用22nm专业制造，内核嵌入式NPU，具有强大的计算及高端图形处理能力，已被广泛应用于智能安防、轻量级AI人工智能、边缘计算、智慧医疗、智能家居、商显等众多领域。

Firefly推出了RK3568全系列智能产品Core-3568J AI核心板、AIO-3568J行业主板。

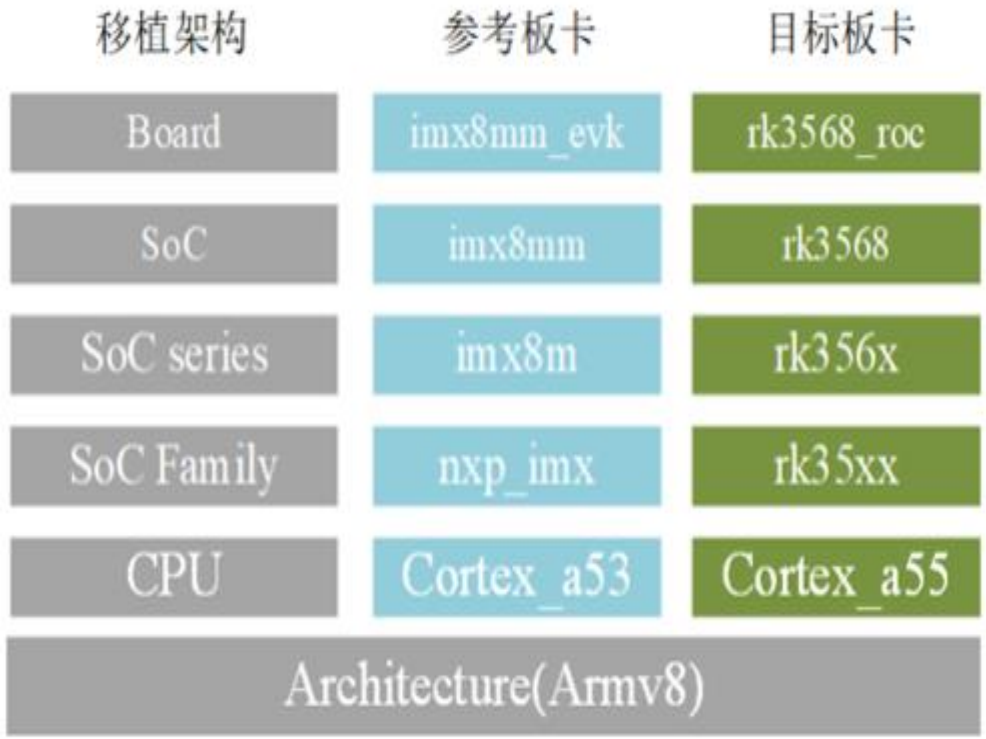
采用Rockchip新一代64位处理器RK3568，集成双核心架构GPU以及高效能NPU；最大支持8G大内存；支持WiFi 6，5G/4G无线传输；拥有丰富的接口扩展，支持多种视频输入输出接口，可用于智能NVR、云终端、物联网网关、工业控制等场景。



Zephyr适配RK3568

整体移植概览

- RK3568在Zephyr上的移植移植整体概览如右图所示,至底向上为架构、CPU、SOC 、BOARD等顺序;
- **ARMv8架构**实现了支持;
- **CPU:**主要是添加4核A55处理器的配置支持。
- **SOC:**外设初始化地址排布说明等配置。
- **BOARD:**包含RK3568硬件平台相关的信息，例如时钟配置、中断定义等。



整体移植逻辑

Zephyr适配RK3568

CPU支持：VHE支持

- **添加CPU相关的配置项目：** 在zephyr/arch/arm64/core/Kconfig文件中需要添加CPU相关的配置项目，这里主要有两个配置项。首先表示使用CORTEX A55的CPU，其次是A55的核拥有ARM处理器的VHE拓展。
- 两个配置的作用：主要用于reset.S文件中初始化过程中的处理器初始化代码，例如使能了HAS_ARM_VHE_EXTN才能使用ARM VHE模式。添加如右图所示：

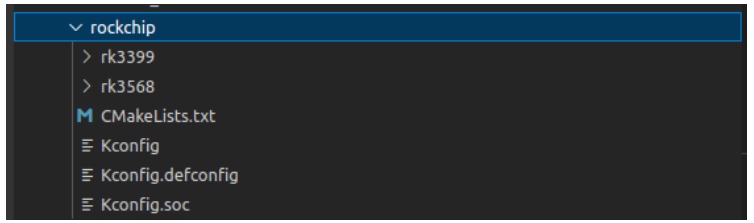
```
config CPU_CORTEX_A53
    bool
    select CPU_CORTEX_A
    select ARMV8_A
    help
      This option signifies the use of a Cortex-A53 CPU

config CPU_CORTEX_A55
    bool
    select CPU_CORTEX_A
    select ARMV8_A
    select ARM64_VHE_MODE
    help
      This option signifies the use of a Cortex-A55 CPU
```

A55 CPU相关配置

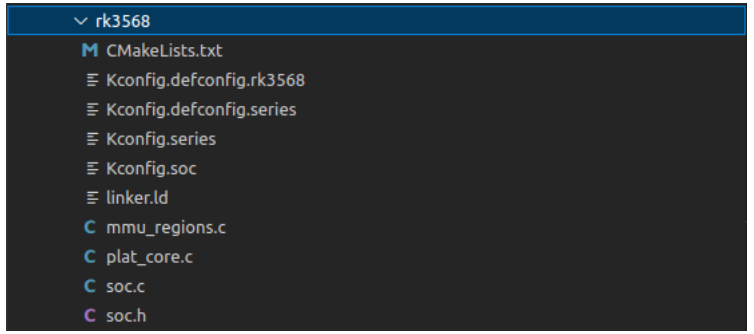
Zephyr适配RK3568

SOC支持：多核启动PSCI

- 在rockchip根目录下，编写的是RK3500Family相关的配置项，如SOC_FAMILY_RK3500的配置项表示是否选择此Family等。此外，在Family的defconfig文件中，需要声明对于.defconfig.series文件的依赖。而在rockchip/rk3568/目录下，主要包含rk3568 series的配置文件以及rk3568 SOC的配置文件。
- 在3个层级中，分别有用于配置项的Kconfig.xxx文件以及作为SoC具体初始化的Kconfig.defconfig.xxx文件。此外，在rockchip/rk3568/目录下还有为SoC支持外设初始化地址排布说明（mmu_regions.c），以及此SoC下应用将使用的链接规则（linker.ld）。



RK3568 SOC



RK3568 SOC

Zephyr适配RK3568

BOARD：基础外设支持

- Zephyr是采用构建阶段生成devicetree.h文件，其中包含设备的宏来使用。设备树和驱动的绑定规则与Linux类似。
- 在dtsi文件中，声明了板卡执行必需的基本外设：如Timer、GIC等等。同时，这些dts文件需要对应相应的.yaml文件以及驱动文件，以实现与系统驱动的绑定，具体如右上图所示。
- 除了.dts和.dtsi文件外，还有board级别的配置文件，规则和SoC命名类似，分别为系统配置项目和板载配置文件。

devicename	compatible	yamlfile	driverfile
gic	arm,gic	arm.gic.yaml	intc_gicv3.c
timer	arm,amrv8-timer	arm,armv8-timer.yaml	arm_arch_timer.c
uart	ns16550	ns16550.yaml	ns16550.c

设备对应文件及驱动

配置文件

Zephyr适配RK3568

Config配置项

- 板载相关基础配置，包括soc,board以及时钟外设；
- 外设相关配置，主要NS16650的串口，以及相关调试程序；
- PSCI的支持，启动多核情况下需要使能，PSCI配置项；
- 优化：只支持非优化版本运行。

zephyr > boards > arm64 > roc_rk3568_pc > M roc_rk3568_pc_defconfig

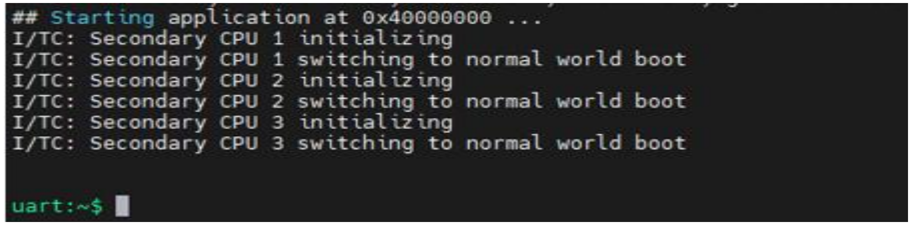
```
1  # SPDX-License-Identifier: Apache-2.0
2  # Platform Configuration
3  CONFIG_SOC_SERIES_RK3568=y
4  CONFIG_SOC_RK3568=y
5  CONFIG_BOARD_ROC_RK3568_PC=y
6  CONFIG_ARM_ARCH_TIMER=y
7
8  # Serial Drivers
9  CONFIG_SERIAL=y
10 CONFIG_UART_NS16550=y
11 CONFIG_UART_INTERRUPT_DRIVEN=y
12 CONFIG_NS16650_EARLYPRINT_DEBUG=n
13 # Enable console
14 CONFIG_CONSOLE=y
15 CONFIG_UART_CONSOLE=y
16
17 # SMP support
18 CONFIG_SMP=y
19 CONFIG_MP_NUM_CPUS=1
20 CONFIG_MP_MAX_NUM_CPUS=1
21 CONFIG_ARMV8_A_NS=y
22 CONFIG_CACHE_MANAGEMENT=y
23
24 CONFIG_PM=y
25 CONFIG_PM_CPU_OPS=y
26 CONFIG_PM_CPU_OPS_PSCI=y
27
28 # 24 MHz system clock, For uart2 1.2Ghz -> 24Mhz -> 1.5Mhz
29 CONFIG_SYS_CLOCK_HW_CYCLES_PER_SEC=24000000
30
31 CONFIG_NO_OPTIMIZATIONS=n
32
```

关键配置项

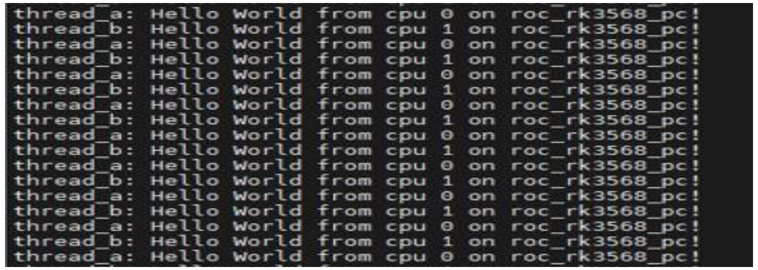
Zephyr适配RK3568

运行示例的结果

- samples/subsys/shell/shell_module。
- samples/synchronization。
- samples/subsys/cpp/cpp_synchronization。
- 其中，shell_module和synchronization能运行成功代表C程序能正常在RK3568上面运行成功。
- samples/subsys/cpp/cpp_synchronization运行成功代表示例项目演示了纯虚拟类（成员）的用法 具有不同类型参数的函数，全局对象构造函数调用。演示了内核的基本健全性。



shell_module



synchronization

cpp_synchronization

CONTENTS

目录

03

基本介绍

Paddle Lite优势特性

➢ 全面通用

- 全面硬件支持：ARM CPU/GPU、Adreno GPU、Apple GPU、NPU、FPGA
- 多操作系统支持：Android、IOS、Linux
- 多训练框架支持：TensorFlow、Caffe、PaddlePaddle、ONNX（支持PyTorch）

动态多目标识别

➢ 轻量实用

- 移动部署，支持模型深度剪裁。 ARM V7（766K） ARM V8（1.2M）

人脸签到

➢ 应用实践模型广泛

- 18个模型benchmark，覆盖图像分类、检测分割和文字识别。
- 80+算子Op支持、85个Kernel。

基本介绍

Paddle Lite架构设计

➢ **模型兼容**

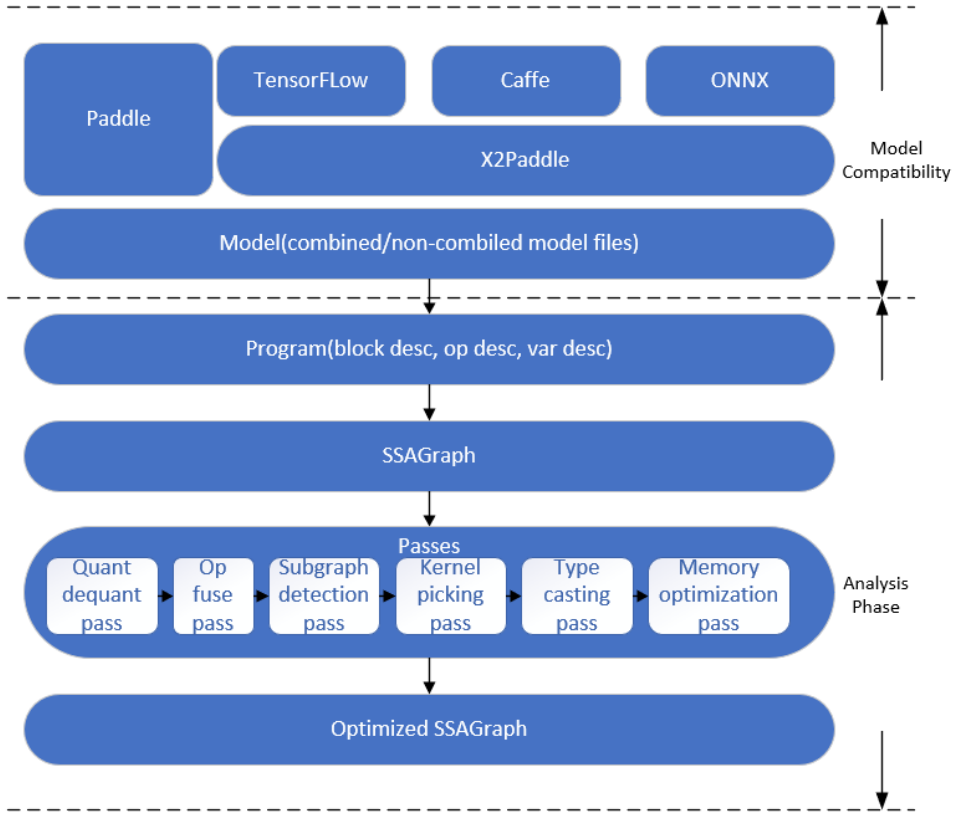
- 对Caffe/TensorFlow/ONNX/PyTorch四大框架的预测模型的转换，PyTorch训练项目的转换。

➢ **分析阶段**

- 包括了 MIR(Machine IR) 相关模块，能够对原有的模型的计算图针对具体的硬件列表进行算子融合、计算裁剪在内的多种优化。

➢ **执行阶段**

- 只涉及到 Kernel 的执行，且可以单独部署，以支持极致的轻量级部署。



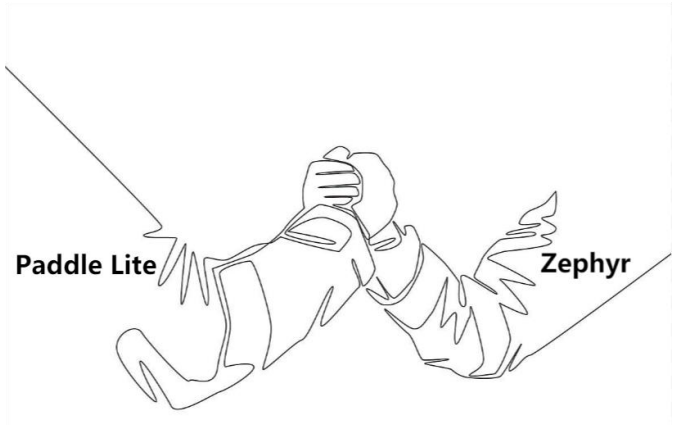
Paddle Lite 架构

基本介绍

融合发展前景

➢ **Paddle Lite**

- 为Paddle Lite端侧深度学习模型部署工具率先适配实时操作系统的支持。
- 解决在资源受限的嵌入设备中进行部署难以满足时序要求的问题。



➢ **Zephyr**

- 为Zephyr在深度学习领域的发展迈出关键一步，使Zephyr使用领域大大扩宽，为之后的发展形成良性循环。

➢ **共同优势**

- 两者都是开源项目，可确保创新方面不受限制。

Paddle Lite2.6适配Zephyr3.4

总体技术路线

- **编译Paddle Lite源码**

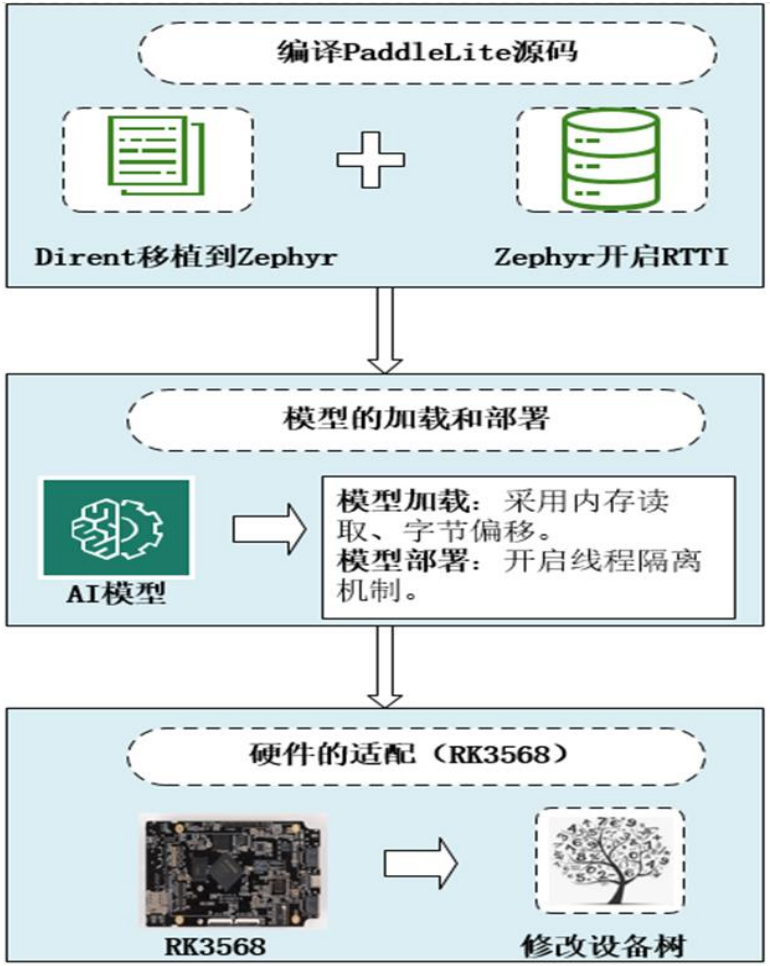
- 添加C++语言标准库的支持，例如，一个是将文件结构体移植到Zephyr（C++库）。另一个是Zephyr开启RTTI功能（C++特性）。

- **模型的加载和部署**

- 模型加载：采用内存读取、字节偏移。模型部署：开启线程隔离机制。

- **硬件的适配（RK3568）**

- 通过修改相关的设备树来适配。



总体路线

Paddle Lite2.6适配Zephyr3.4

Zephyr Paddle Lite 部署思路

➢ **源代码文件**

- Paddle Lite作为一个模块加入Zephyr项目中，并编写测试sample。具体来说：通过ldd命令查看Paddle Lite生成静态库包含的200多个文件，将200多个文件放入Zephyr形成一个module。

➢ **编译修改支持**

- 修改Cmake文件，在系统构建过程中添加对Paddle lite文件的支持。同时修改SDK中编译工具，使之支持Paddle Lite功能。

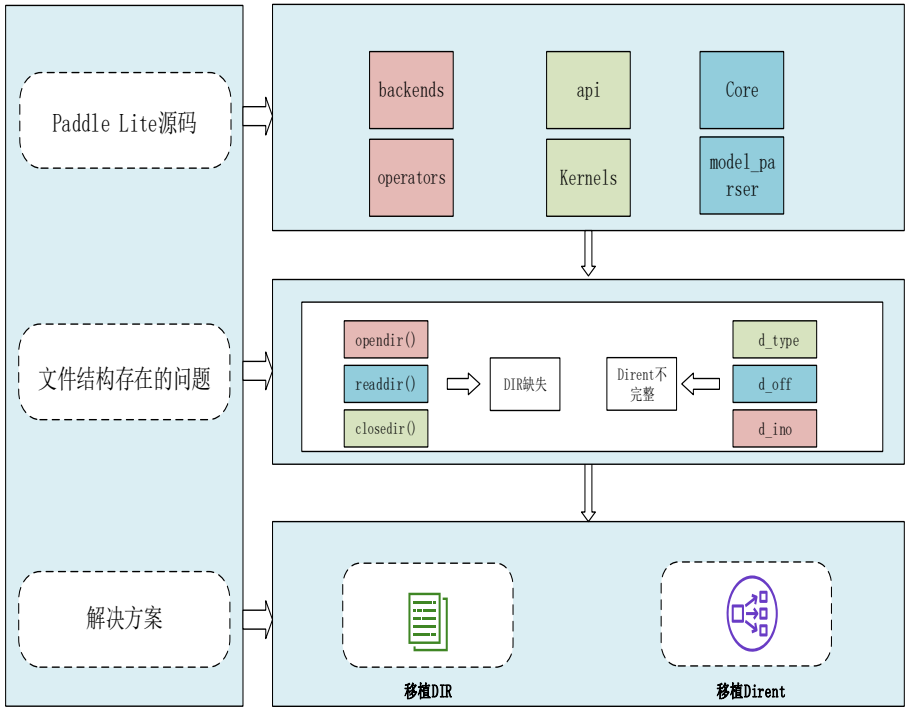
➢ **配置项支持**

- 修改配置文件，使能Paddle Lite运行时需要的第三方模块支持（如C++拓展）。具体来说：将Paddle Lite可能用到的模块，全部在prj.config文件开启相关选项。

Paddle Lite2.6适配Zephyr3.4

文件结构体问题

- **dirent.h是一个标准C库头文件**，用于在Unix、Linux等操作系统上访问目录中的文件和子目录。该文件定义了一些结构体和函数，例如dirent结构体，以及opendir()、readdir()、closedir()等函数，可用于打开、读取和关闭目录，并查看其内容。
- 在Zephyr中，dirent结构体不完整是因为Zephyr使用了不同于UNIX的文件系统方式。为了适应这种不同的文件系统实现，Zephyr的dirent结构体只定义了文件名和文件类型等基本信息，而没有包含其他如inode、文件偏移量等信息。需要补充完整结构体信息。流程图如右图所示。



文件结构体解决流程

Paddle Lite2.6适配Zephyr3.4

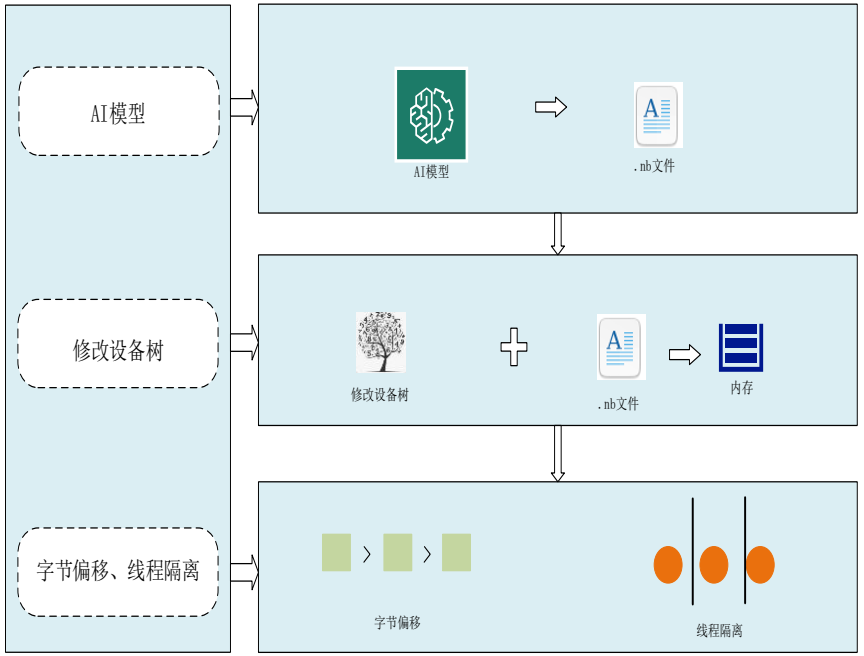
RTTI问题

- RTTI 全称为 Run-Time Type Information，即运行时类型信息。它是 C++ 的一个特性，提供了在运行时获取对象类型信息的支持。RTTI 机制主要包含两个运算符：typeid 运算符和dynamic_cast 运算符。typeid 运算符获得表达式类型的对象，返回一个 std::type_info 类型的对象。 dynamic_cast 运算符用于将一个基类指针或引用转换为其派生类的指针或引用。
- Zephyr中关闭了RTTI 的功能，需要将所有的RTTI功能进行开启。为了开启RTTI，需要修改编译器的编译选项，以支持RTTI功能。具体而言，需要将编译选项中的-fno-rtti改为-frtti，以启用RTTI功能。具体如下：
- 1.打开Zephyr应用程序的CMakeLists.txt文件，找到目标app。
- 2.将no-rtti选项替换为rtti，例如：target_compile_options(app PRIVATE -frtti)
- 3. 重新编译应用程序，以使修改生效。

Paddle Lite2.6适配Zephyr3.4

基于内存的文件访问

- Zephyr 是一个面向嵌入式系统的开源实时操作系统（RTOS）。它的设计目标是运行在资源有限的嵌入式设备上，因此在某些情况下可能没有内置的文件系统。资源受限的嵌入式设备通常具有有限的存储空间和处理能力。
- 首先将AI的模型文件通过Paddle Lite的opt工具变成Paddle Lite所认识的模型文件。采取将模型文件放入内存，通过读内存以及字节偏移的方式来读取模型文件。修改对应的设备树文件，并且完成物理地址到虚拟地址的映射。同时需要开启线程隔离机制，防止一个任务因为线程资源的耗尽而影响到其他任务。



模型读取流程图

Paddle Lite2.6适配Zephyr3.4

prj.config文件

- Zephyr 中，prj.config 文件是一个项目配置文件，用于指定构建应用程序时的各种选项和参数。它是一个文本文件，包含一系列键值对，用于配置 Zephyr 的各种功能和行为。对于运行Paddle Lite应用需要的配置项如右图所示。
- CONFIG_CPLUSPLUS：开启C++支持。
- CONFIG_PADDLE_LITE：开启Paddle Lite功能。
- CONFIG_THREAD_LOCAL_STORAGE：在内核中启用线程本地存储 （TLS） 支持。
- CONFIG_KERNEL_VM_SIZE：内核地址空间的大小。
- CONFIG_MAX_XLAT_TABLES：指定转换表的最大数量。

```
CONFIG_CPLUSPLUS=y
CONFIG_PADDLE_LITE=y
CONFIG_THREAD_LOCAL_STORAGE=y
CONFIG_KERNEL_VM_SIZE=0x10000000
CONFIG_THREAD_STACK_INFO=y
CONFIG_ASSERT=y
CONFIG_MAX_XLAT_TABLES=56
CONFIG_TIMING_FUNCTIONS=y
```

prj.config

Zephyr+RK3568+Paddle Lite示例

图像分类简介

- 图像分类是一种计算机视觉任务，其目标是将输入的图像分成不同的类别。在图像分类任务中，计算机接收一张输入图像，经过预处理后输出一个代表该图像所属类别的预测结果。

- 例如，我们假定一个可能的类别集categories = {dog, cat}，之后我们提供一张图片（右图所示）给分类的模型：这里的目标是根据输入图像，从类别集中分配一个类别。

图像分类

Zephyr+RK3568+Paddle Lite示例

Paddle Lite模型

- 目前，Paddle Lite 已严格验证 52 个模型的精度和性能。对视觉类模型做到了充分的支持，覆盖分类、检测和定位，也包含了特色的 OCR 模型的支持。对 NLP 模型也做到了广泛支持，包含翻译、语义表达等等。

- 除了已严格验证的模型，Paddle Lite 对其他 CV 和 NLP 模型也可以做到大概率支持。

类别	类别细分	模型	支持平台
CV	分类	MobileNetV1	ARM, X86, GPU(OPENCL,METAL), HuaweiKirinNPU, RockchipNPU, MediatekAPU, KunlunxinXPU, HuaweiAscendNPU, VerisiliconTIMVX, AndroidNNAPI
CV	分类	MobileNetV2	ARM, X86, GPU(OPENCL,METAL), HuaweiKirinNPU, KunlunxinXPU, HuaweiAscendNPU
CV	分类	MobileNetV3_large	ARM, X86, GPU(OPENCL,METAL), HuaweiAscendNPU
CV	分类	MobileNetV3_small	ARM, X86, GPU(OPENCL,METAL), HuaweiAscendNPU
CV	分类	DPN68	ARM, X86, HuaweiAscendNPU
CV	分类	AlexNet	ARM, X86, HuaweiAscendNPU
CV	分类	DarkNet53	ARM, X86, HuaweiAscendNPU
CV	分类	DenseNet121	ARM, X86, HuaweiAscendNPU
CV	分类	EfficientNetB0	ARM, X86, GPU(OPENCL), KunlunxinXPU, HuaweiAscendNPU
CV	分类	GhostNet_x1_3	ARM,HuaweiAscendNPU
CV	分类	HRNet_W18_C	ARM, X86,HuaweiAscendNPU
CV	分类	RegNetX_4GF	ARM, X86
CV	分类	Xception41	ARM, X86
CV	分类	ResNet18	ARM, X86, GPU(OPENCL,METAL), HuaweiKirinNPU, RockchipNPU, KunlunxinXPU, HuaweiAscendNPU, VerisiliconTIMVX, AndroidNNAPI
CV	分类	ResNet50	ARM, X86, GPU(OPENCL,METAL), HuaweiKirinNPU, RockchipNPU, KunlunxinXPU, HuaweiAscendNPU, VerisiliconTIMVX, AndroidNNAPI, IntelOpenVINO
CV	分类	ResNet101	ARM, X86, HuaweiKirinNPU, RockchipNPU, KunlunxinXPU, HuaweiAscendNPU
CV	分类	ResNeXt50	ARM, X86, HuaweiAscendNPU
CV	分类	MnasNet	ARM, HuaweiKirinNPU, HuaweiAscendNPU
CV	分类	SqueezeNet	ARM, HuaweiKirinNPU, KunlunxinXPU, HuaweiAscendNPU
CV	分类	ShuffleNet	ARM,HuaweiAscendNPU
CV	分类	ShufflenetV2	ARM, KunlunxinXPU, HuaweiAscendNPU

Paddle Lite提供的模型

Zephyr+RK3568+Paddle Lite示例

Paddle Lite 推理流程

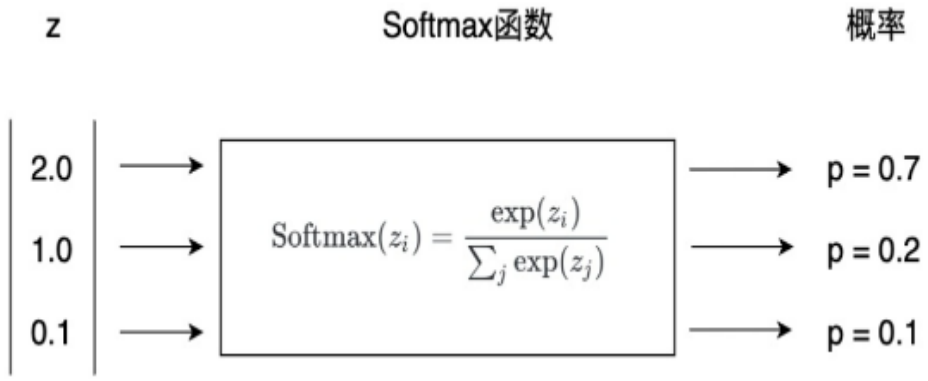
- 右图是Paddle Lite进行模型推理的流程。
- 配置 Config 信息：用于在模型部署和推理过程中对 Paddle Lite 的运行配置。
- 加载模型：可以使用 PaddlePaddle 框架将训练好的模型转换为 Paddle Lite 支持的格式（如.nb格式）。
- 创建 Predictor 对象：完成模型解析和环境初始化，并获取输入和输出 tensor 的维度信息。
- 输入数据：输入数据的格式要与转换为 Paddle Lite 支持格式的模型相匹配。
- 执行推理：使用 Predictor 对象的成员函数 Run()。
- 输出数据：获取输出 tensor。

Paddle Lite推理流程

Zephyr+RK3568+Paddle Lite示例

输入和输出

- 输入：在计算机中视角下的图片通常以数字形式表示，一张图片在Paddle Lite将被表示为一个张量（tensor），它是一个高维度的矩阵。我们把这个高维度矩阵中的所有数值都设置为1，来模拟一张图片。（后续的demo中都是全1作为输入来模拟一张图片）
- 输出：在图像分类任务中，最后是经过一层全连接层（fc）加上Softmax函数，输出的是概率信息。例如，对于1000分类，那么最后是经过fc=1000的全连接层加上Softmax函数，并且输出1000个概率，在1000个概率中最大的那个就是对应的那个图像的分类。右图是sofxmax函数的图解。



Softmax函数

Zephyr+RK3568+Paddle Lite示例

Mobilenet_v1模型

- Mobilenet_v1是谷歌在2017年提出的，专注于移动端或者嵌入式设备中的轻量级CNN网络。该论文最大的创新点是，提出了深度可分离卷积。v1的整个模型结构如右上图所示，该网络有28层（fc算在内，pool和softmax不算，层数计算只考虑有参数的层）。相比传统卷积神经网络，在准确率小幅降低的前提下大大减少模型参数与运算量。(相比VGG16准确率减少了0.9%，但模型参数只有VGG的1/32)。
- 在Zephyr3.4+Paddle Lite2.6+RK3568下运行的结果如右下图所示。输出为1000个概率，选取了其中10个作为输出。

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1 Conv / s1	3 × 3 × 512 dw 1 × 1 × 512 × 512	14 × 14 × 512 14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

Mobilenet_v1

```
*** Booting Zephyr OS build zephyr-v3.4.0-1177-gc607179cc46d ***
Output shape 1000
Output[0]: 0.000191311
Output[100]: 0.000159713
Output[200]: 0.000264313
Output[300]: 0.000210793
Output[400]: 0.00103236
Output[500]: 0.000110071
Output[600]: 0.00482922
Output[700]: 0.00184533
Output[800]: 0.000202115
Output[900]: 0.000585589
```

输出结果

Zephyr下推理时间为0.13秒，Linux下为0.42秒

Zephyr+RK3568+Paddle Lite示例

Mobilenet_v2模型

- Mobilenet_v2是由谷歌团队在2018年提出的，相比Mobilenet_v1网络，准确率更高，模型更小。该架构提高了移动模型在多个任务和多个基准数据集上以及在不同模型尺寸范围内的最佳性能。模型的网络结构如右上图所示。
- 在Zephyr3.4+Paddle Lite2.6+RK3568下运行的结果如右下图所示。输出为1000个概率，选取了其中10个作为输出。

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Mobilenet_v2

```
*** Booting Zephyr OS build zephyr-v3.4.0-1177-gc607179cc46d ***
Output shape 1000
Output[0]: 0.000227615
Output[100]: 0.000920521
Output[200]: 0.000418927
Output[300]: 0.000894782
Output[400]: 0.000757932
Output[500]: 0.000103452
Output[600]: 0.00596757
Output[700]: 0.000948417
Output[800]: 3.49579e-05
Output[900]: 0.00256156
```

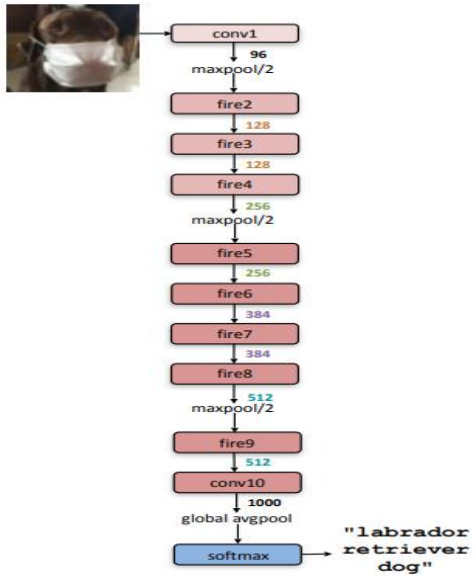
输出结果

Zephyr下推理时间为0.11秒，Linux下为0.44秒

Zephyr+RK3568+Paddle Lite示例

SqueezeNet模型

- 大多数神经网络追求提升识别精确率为主要目的，采用手段的主要方向包括加深网络结构和增强卷积模块功能，同时这也导致了整个用于识别的网络越来越复杂，需要的内存和算力也大大增加。SqueezeNet开辟了另外一个方向：在保证模型精度不降低的前提下，最大程度的提高运算速度。它能够在ImageNet数据集上达到近似AlexNet的效果，但是参数量相比AlexNet少了50倍，网络结构如图右上图所示。
- 在Zephyr3.4+Paddle Lite2.6+RK3568下运行的结果如右下图所示。输出为1000个概率，选取了其中10个作为输出。



SqueezeNet

```
*** Booting Zephyr OS build zephyr-v3.4.0-1177-gc607179cc46d ***
Output shape 1000
Output[0]: 0.000139096
Output[100]: 0.000117584
Output[200]: 0.000177571
Output[300]: 0.00108058
Output[400]: 0.000798813
Output[500]: 8.10145e-05
Output[600]: 0.0112027
Output[700]: 0.00258788
Output[800]: 0.000215435
Output[900]: 0.000332555
```

输出结果

Zephyr下推理时间为0.061秒，Linux下为0.38秒

CONTENTS

目录

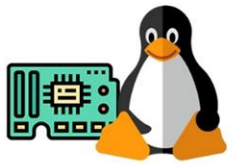
04

基本介绍

高性能嵌入式应用场景：万物互联

- **嵌入式设备性能提升** (MCU->SoC、ARM 64)
- **嵌入式系统功能丰富**（富功能+实时控制）
- **嵌入式系统开发度增加**（智能物联网时代）

面向新一代万物互联的分布式嵌入式场景



基本介绍

混合关键部署需求

- 当前分布式嵌入式场景存在富功能（数据可视化、场景可视化）与硬实时（精准控制）的双重需求

在单个系统上打造双子星（富功能OS，实时OS）互助运转的混合关键部署模式

嵌入式OS对富功能与硬实时的混合关键部署要求

双子星联动

富功能OS

- 智能座舱
- 数据可视化
- 场景可视化

Linux宏内核

Linux：功能强大但实时性不足

实时控制OS

- 智能驾驶
- 底盘控制
- 动力控制

实时微内核

QNX：以闭源的方式来形成技术壁垒

满足关键场景对智能化、实时性及安全性的多重要求！

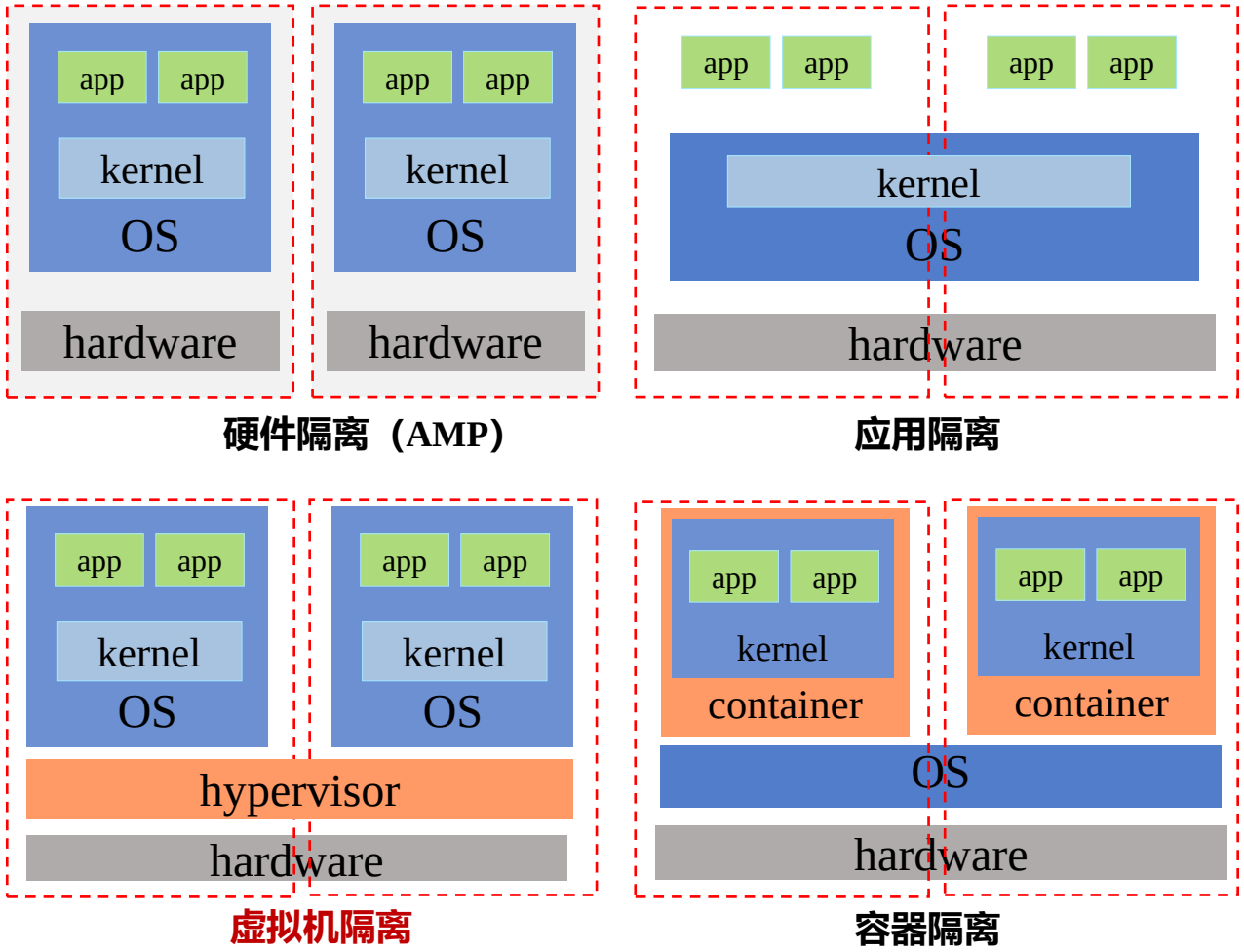
基本介绍

安全隔离保护机制

- **解耦双子星故障联动，保证双子星彼此隔离与保护**

- 硬件隔离：在硬件上直接为OS隔离资源
- 应用隔离：在操作系统级别实现应用的隔离
- 虚拟机隔离：在虚拟化层实现应用隔离
- 容器隔离：通过容器技术实现应用的隔离

本项目使用虚拟化技术（Hypervisor）实现隔离

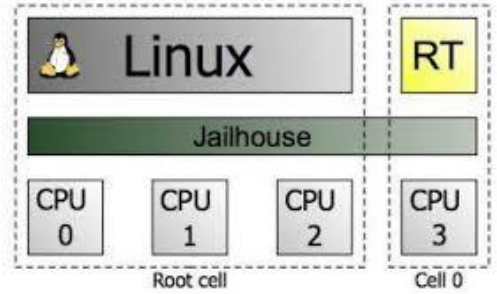


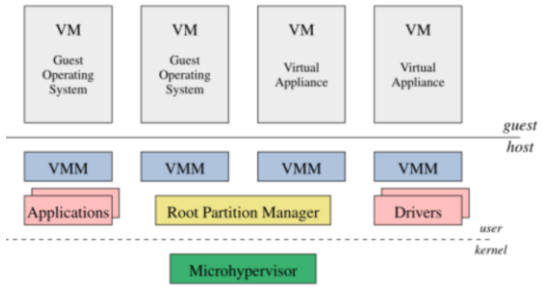
基本介绍

常见Hypervisor方案

- **开源虚拟化方案**

  主要包括XEN, ACRN， Xvisor, Linux KVM等为代表；



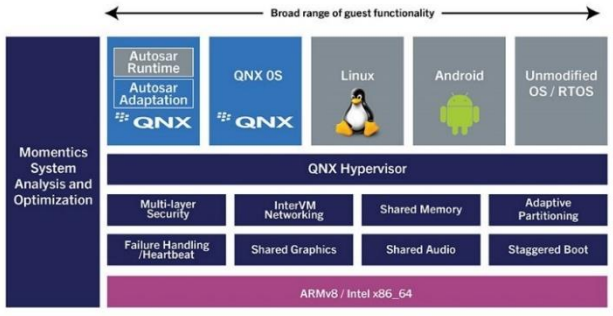


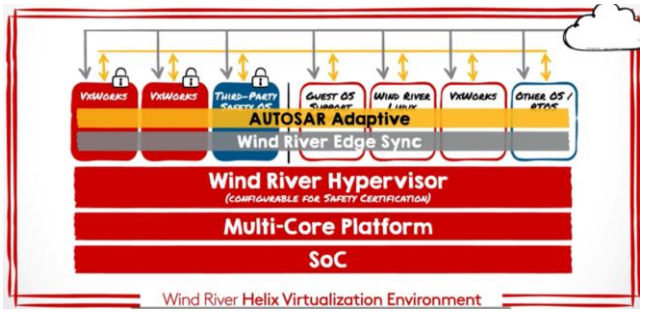
基本介绍

常见Hypervisor方案

- **商业闭源虚拟化解决方案**

主要包括QNX Hypervisor，Wind River Helix，PikeOS，VMware vSphere等商业解决方案

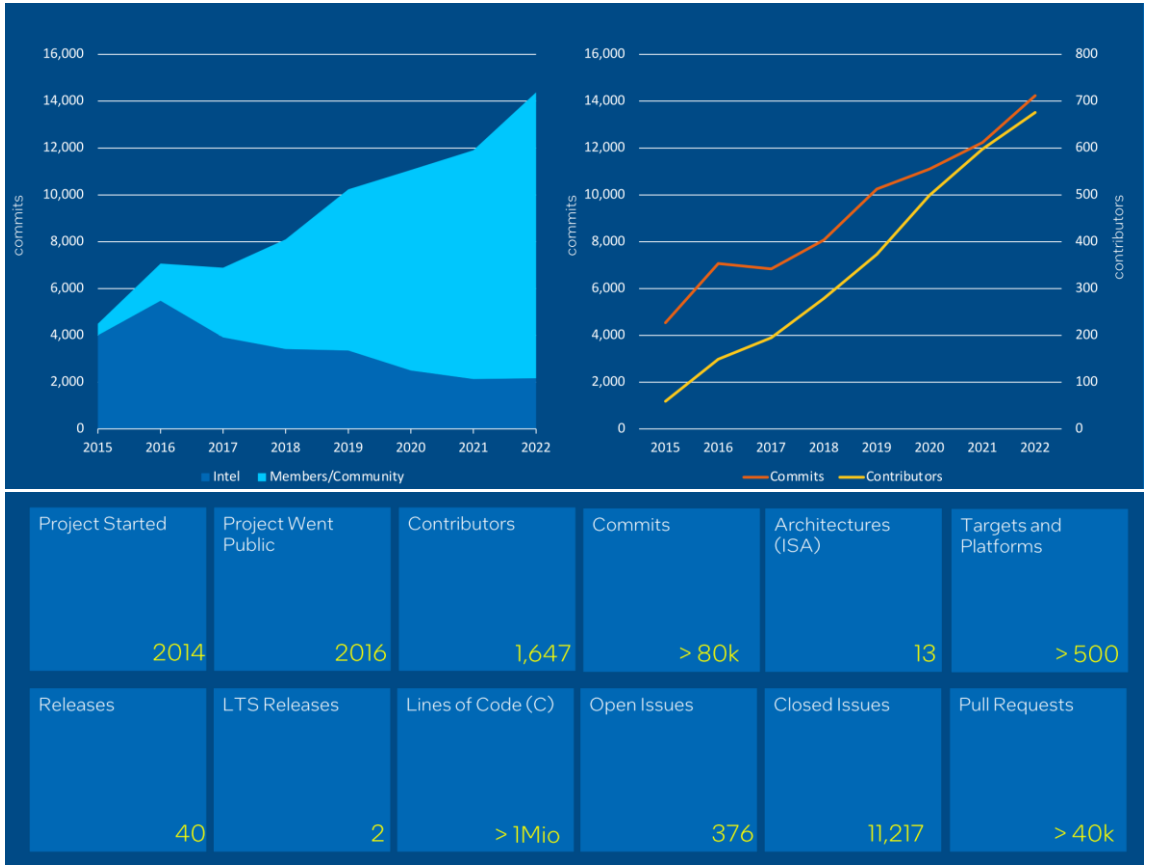




Zephyr-Based Virtual Machine

ZVM优势：基于开源实时操作系统Zephyr

- 活跃的开源社区，**开源贡献者超过1600个、提交数量达到80k+**；
- 支持不同硬件，超过**13种硬件架构**；
- 支持超过**500+的实际板卡**；
- Zephyr是一个产品级应用，发布了**40个release**版本，包含**2个LTS版本**；
- 良好的**实时性和安全性**支持；
- 支持多种嵌入式场景下的**物联网协议。**



Zephyr 开源项目数据（2023）

Zephyr-Based Virtual Machine

ZVM优势：基于新一代硬件特性（VHE）

➢ **nVHE模式**

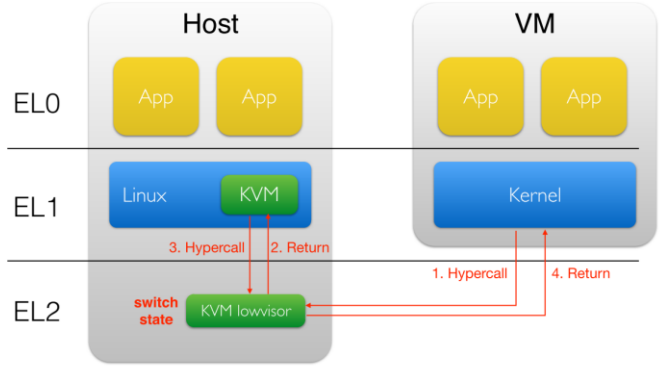
- nVHE模式下，主机和客户机内核都运行于EL1特权级，而主机中Hypervisor模块运行于EL2层。

➢ **VHE特性**

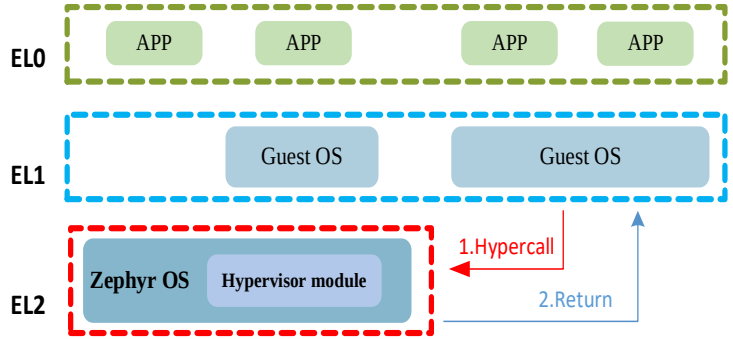
- 支持主机内核可以在EL2上执行。

优势：基于**ARMv8.1**新一代高性能架构(**RK356X系列**)，减少了上下文切换次数，**提高系统性能**；同时简化Hypervisor系统架构，**系统更加简洁高效。**

新架构芯片及SoC



nVHE模式架构（开销高）



VHE模式架构（开销低）

Zephyr-Based Virtual Machine

ZVM优势：系统灵活的可配置性

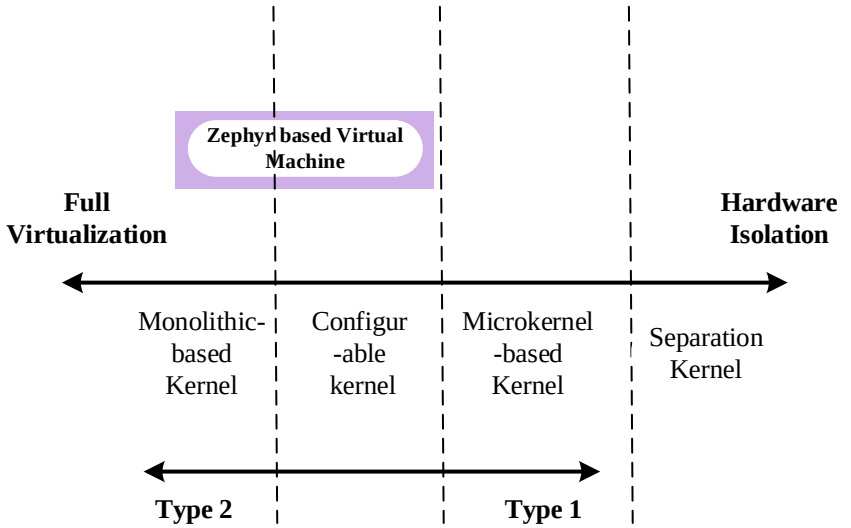
在ZVM系统中，内核、子系统、驱动程序及用户代码都可以根据场景需求进行配置。

➢ **依赖Zephyr配置系统的特性**

- 支持交互式Kconfig界面，包括menuconfig核guiconfig；
- 支持west工具自动化构建与运行；
- 支持完善的Kbuild和Cmake构建系统；

➢ **ZVM虚拟化系统可配置**

- Zephyr-based Virtual Machine跨越**可配置内核到宏内核**；
- ZVM最小系统配置具有**类似微内核特性**；
- ZVM完全功能配置具有**宏内核特性**；



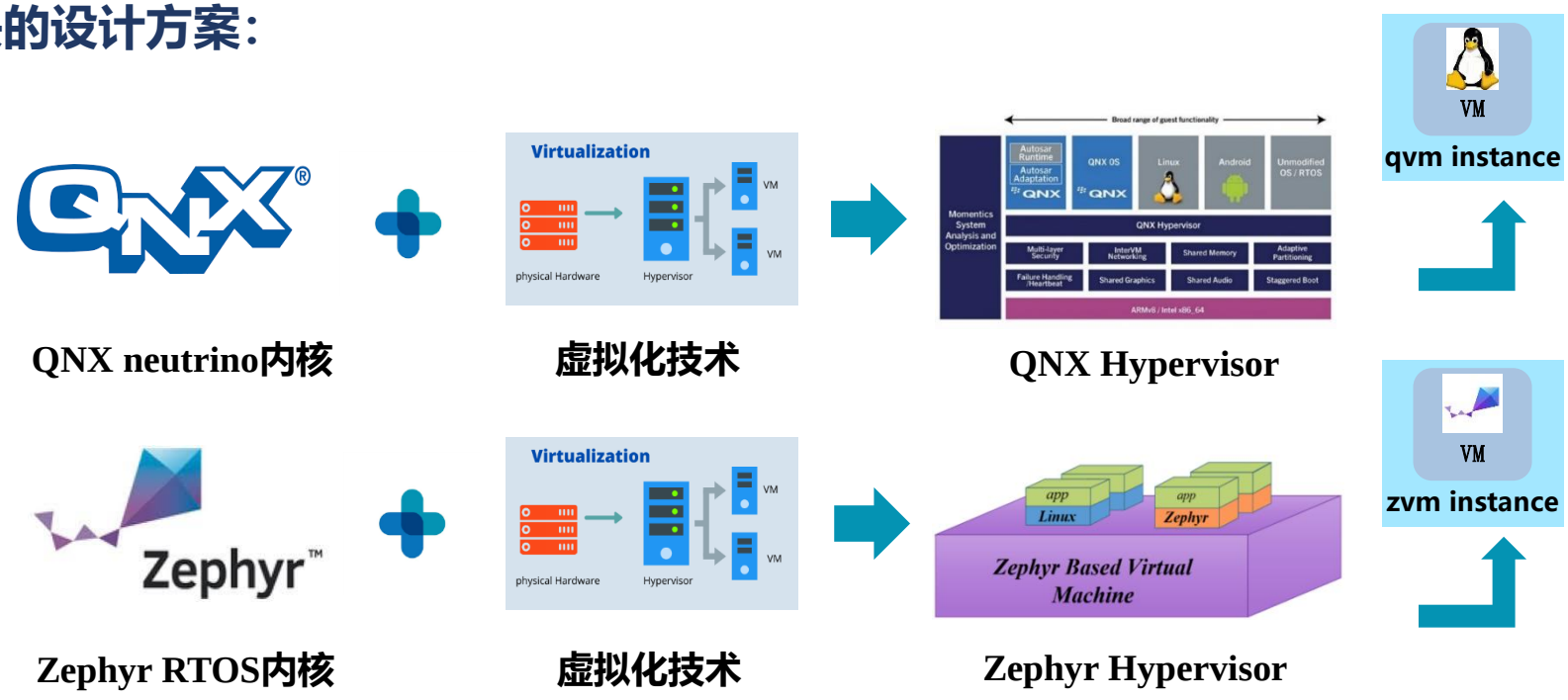
ZVM虚拟化系统可配置

Zephyr-Based Virtual Machine

ZVM优势：基于Zephyr内核的拓展虚拟机管理器

参考QNX的内核 + 虚拟化模块的设计方案：

- Zephyr **可裁剪的内核**结合**虚拟化技术**，实现类QNX hypervisor的虚拟化解决方案。
- 每一个**ZVM instance**管理一个虚拟机实例，并为其分配资源



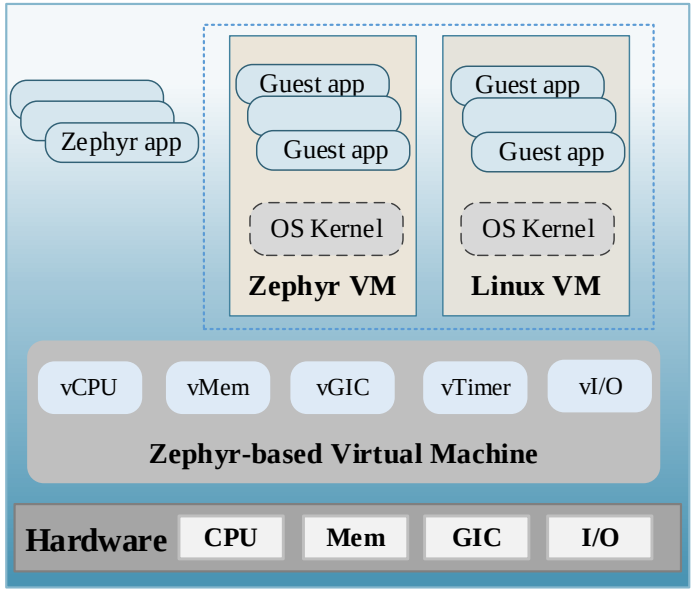
目标：开源、实时，安全，可靠

整体方案设计

五大模块设计

➢ **主要功能模块设计与实现：**

- 虚拟处理器（vCPU）模块： 核心计算资源
- 虚拟内存（vMem）模块： 核心存储资源
- 虚拟中断（vGIC）模块： 物理外设中断+软件模拟中断
- 虚拟设备（vDev）模块： 完全虚拟化+直通
- 虚拟时钟（vTimer）模块： Host记录VM的事件



系统架构图

Zephyr RTOS提供线程调度器、内存管理单元、中断管理模块、定时器、设备管理模块等支持

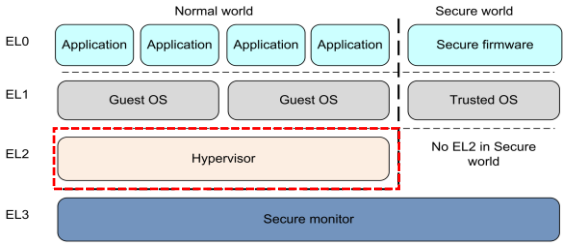
硬件辅助虚拟化

ARMv8架构硬件虚拟化支持

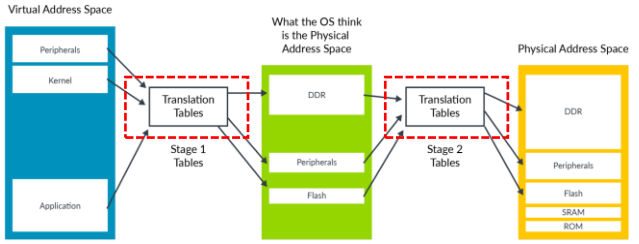
➢ **完善的硬件虚拟化支持**

- **处理器虚拟化拓展支持**，为Hypervisor设计了单独的特权级（**EL2 mode**）
- **内存虚拟化拓展支持**，支持两阶段物理地址转换(**Stage-2 Translation**)
- **中断虚拟化拓展支持**，提供一组物理寄存器支持虚拟中断注入(**vCPU Interface**)
- **外设虚拟化拓展支持**，为DMA设备提供单独硬件地址转换机制(**SMMU**)

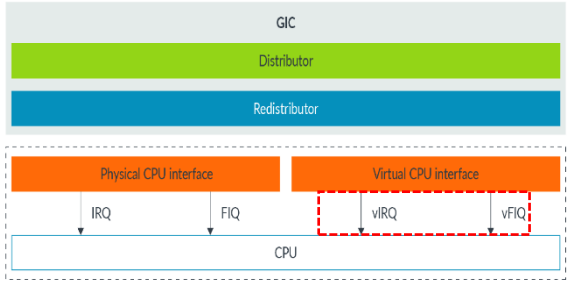
显著降低系统虚拟化开销



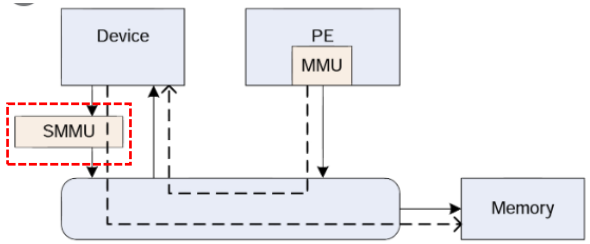
Hypervisor mode



Stage-2 Translation



vCPU Interface



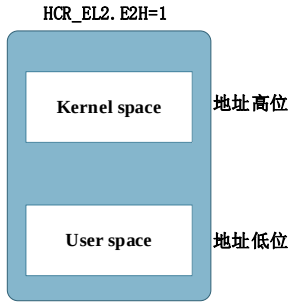
SMMU Translation

硬件辅助虚拟化

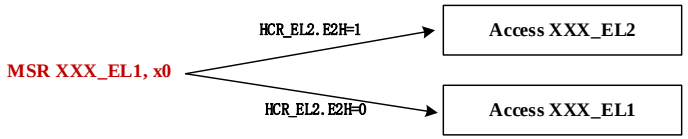
ARMv8虚拟化主机拓展支持

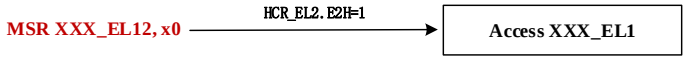
➢ Virtualization Host Extension（VHE）

- **Host OS直接运行于EL2 mode，显著减少上下文切换开销**
- **EL2内存模型实现用户态/内核态内存访问**
- **TLB中增加ASID，支持MMU快速地址转换**
- **EL1寄存器自动重定向为EL2寄存器，避免修改主机内核**
- **增加EL1别名寄存器，支持Hypervisor控制虚拟机**



EL2内存模型





寄存器别名访问　　寄存器重定向

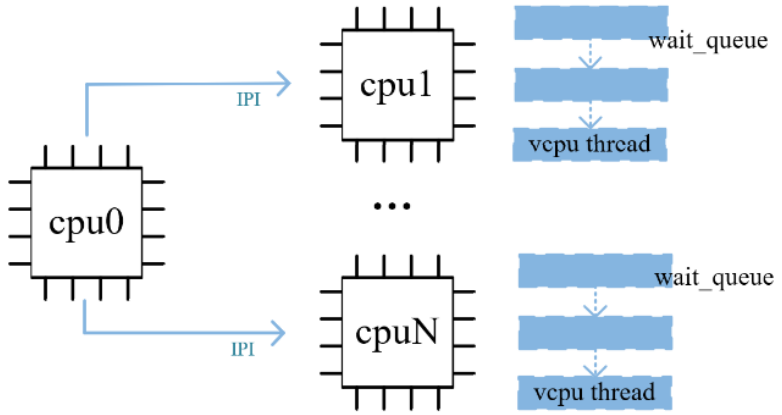
- VHE模式提供了ttbr0_el2和ttbr1_el2两个寄存器。解除了以往非VHE EL2模式下不能访问用户空间的限制。
- ASID（地址空间标识符）标记应用（减少不必要的切换）以减少开销。MMU在做地址转换时会将TLB表项里的ASID和当前进程的ASID值做比较，只有ASID值相等，MMU才认为这条表项是我需要的。
- 非VHE模式下，主机如果要运行在EL2模式，需要将EL1模式下访问的以EL1后缀结尾的寄存器修改为以EL2结尾的寄存器（ARM架构以不同的后缀标识不同访问级别）。而在VHE模式下，对一些访问EL1后缀的寄存器，硬件自动变为访问EL2，所以不需要修改内核代码。
- 因为第3点的存在，原先处在Hypervisor模式下访问EL1寄存器时都变为了访问EL2寄存器。为了解决这个问题，ARM硬件新增加了一些别名寄存器，专门为VHE模式下的Hypervisor访问虚拟机寄存器而使用，非VHE模式下基本不用。

虚拟处理器模块设计

vCPU配置方法

➢ **采用主从核通信：**

- **cpu0负责主机的任务调度**，如shell输入产生的中断将路由至cpu0进行处理，执行控制指令。
- cpu0与cpuN间的通信通过核间中断(Inter-Processor Interrupt,IPI)方式实现，cpu0通过IPI通知cpuN执行任务。
- **vCPU线程在初始化过程中绑定一个物理CPU**，并在启动时部署到指定cpu上执行，直到异常发生。



SMP系统核间通信

虚拟内存模块设计

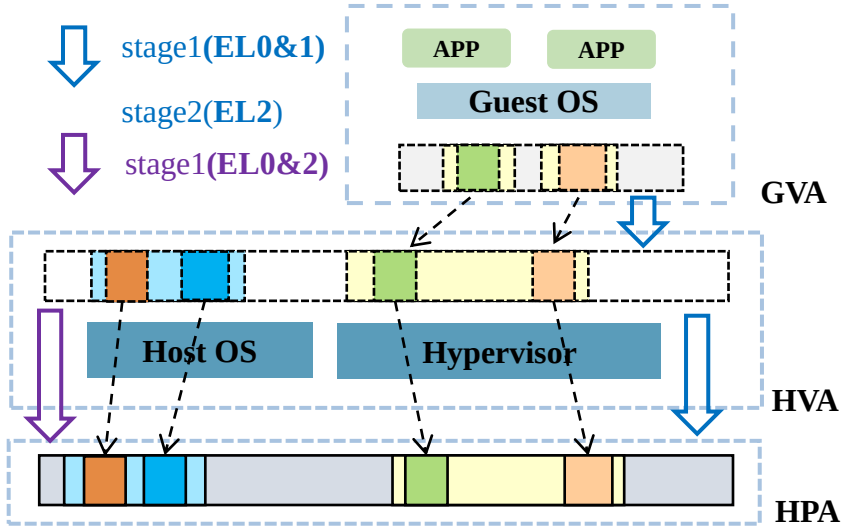
内存地址转换

- **内存虚拟化：**实现VM内存地址的隔离，并监控VM对实际物理内存的访问，以实现对主机物理内存的保护

- **主机操作系统stage-1转换：**主机操作系统经stage-1地址转换，直接由HVA经过一次地址转换找到HPA

- **客户机操作系统stage-2转换**

- 客户机操作系统通过自身地址映射，实现stage-1地址转换，由GVA找到HVA
- Hypervisor通过stage-2地址转换机制，由HVA转换为真实物理地址HPA



虚拟化内存地址转换

内存虚拟化的2个关键技术： 1）vMem的构造；2）vMem地址映射

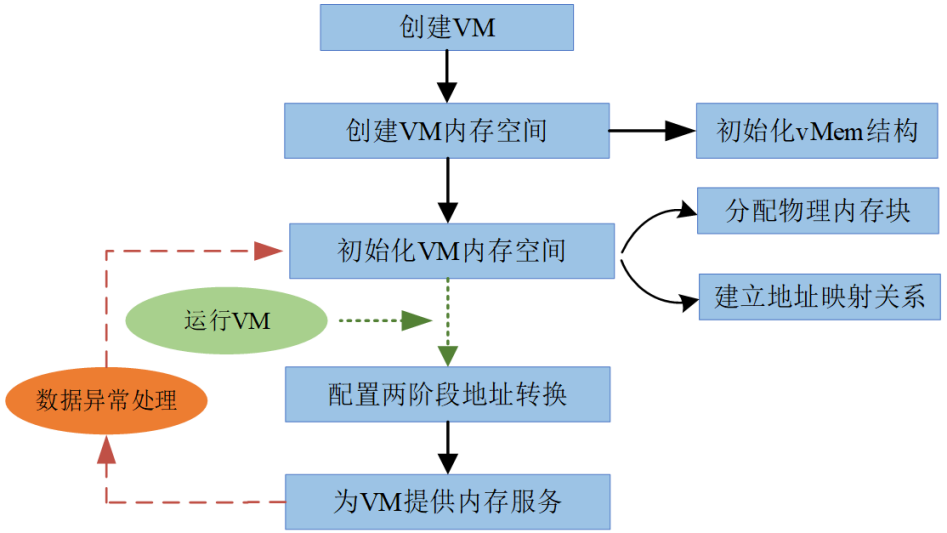
虚拟内存模块设计

vMem模块实现

- **此模块主要分为如下两个部分：**

- 创建VM时根据参数初始化vMem结构体，将虚拟地址空间以Memory Area进行划分；所有Area空间组成的链表，共同构成VM内存空间。

- 为Memory Area分配物理内存并加载OS镜像文件，加载完成后建立地址映射关系，以配置两阶段的地址转换能力，最终为VM提供内存服务。



vMem执行流程

虚拟中断模块设计

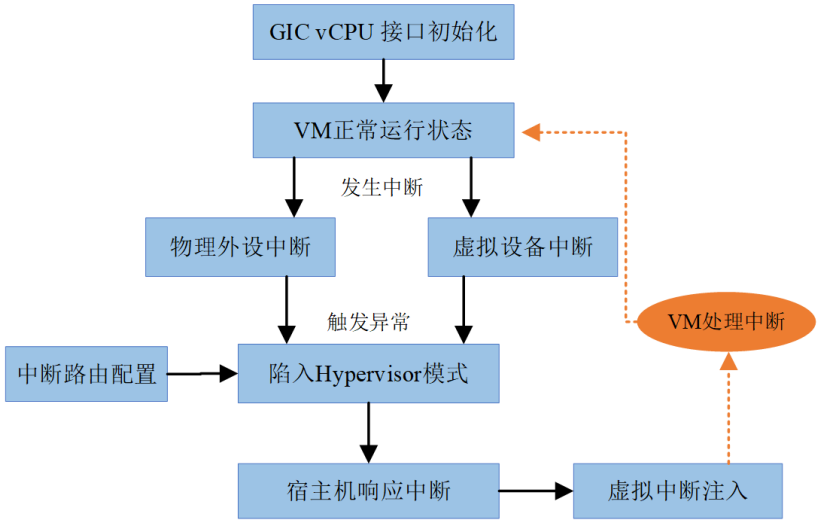
vGIC模块实现

- 中断虚拟化：使用ARM GIC设备，并基于其实现虚拟中断的配置；通过GIC中的Virtual CPU interface和List Register（物理CPU中有一些List register专门用作存储虚拟中断信息）来实现虚拟中断的注入。vGIC就是为VM模拟了一个GIC控制器，然后Hypervisor监控VM对物理GIC的操作。

- **此模块主要分为如下两个部分：**

  - **配置vGIC接口**，并将需发送给VM的虚拟中断统一存储到Pend_list链表中，构成当前VM的需处理虚拟中断。list register的数量有限(最多16个，现在平台上默认只有4个，而真实系统中irq号可能有几十上百个)，通过Pend_list作为等待list

  - **中断注入：**在VM Entry前，根据Pend_list中的中断数据配置相应虚拟中断接口实现中断注入，进入VM即可响应并处理中断。



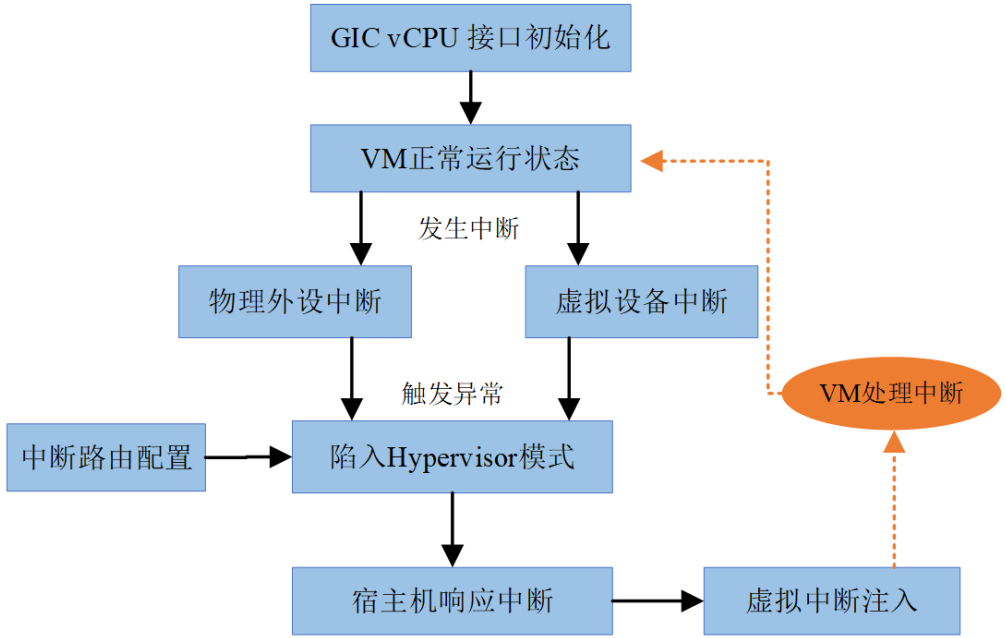
vGIC执行流图

虚拟中断模块设计

实现的中断类型

➢ **现阶段支持的中断处理类型有两种：**

- **物理外设中断**（如串口产生的中断）。这里我们将其统一路由到EL2特权级并由Hypervisor进行处理。
- **软件模拟中断**（vCPU间的核间中断）。这类中断用于模拟核间中断IPI（Inter-Processor Interrupt），并由此使得主机能够控制VM；IPI中断首先仍然也是由Hypervisor接管，并根据中断处理函数进行相应处理（因为现阶段VM是单核环境，所以通过主机CPU给VM vCPU发送核间中断）。



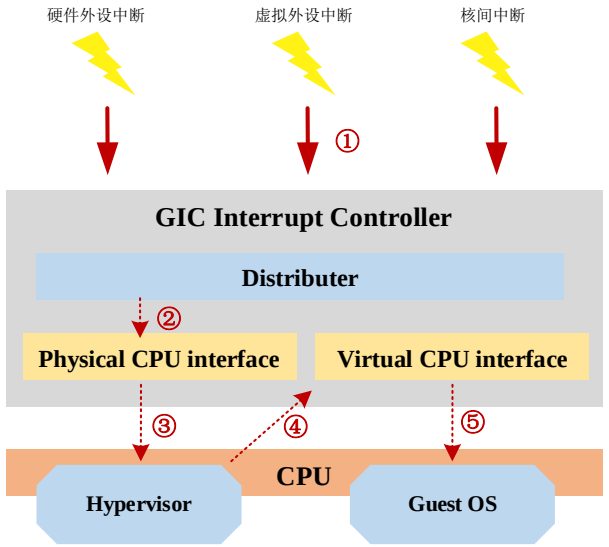
vGIC执行流图

中断虚拟化的2个关键技术： 1）vGIC的构造；2）中断处理流程的实现

虚拟中断模块设计

虚拟中断处理流程

- 外设产生一个中断发送到Distributor
- Distributor把这个中断发送给Physical CPU interface
- Physical CPU interface告诉Hypervisor去处理这个中断
- Hypervisor对这个中断进行检查，发现这个中断是送给VM处理的，它会设置一个对应的虚拟中断，把这个虚拟中断加入到Virtual CPU interface
- Virtual CPU interface会根据Hypervisor加入的虚拟中断， 向List register写入内容（用作存储虚拟中断信息），然后VM就可以感知中断是给它的
- VM通过virtual CPU interface发来的中断进行处理(在EL1模式)，处理后返回
- Virtual CPU interface发现这个虚拟中断来自于一个物理中断，就会在Dirtributor上清除这个物理中断（表示处理完毕），整个虚拟中断处理过程结束



虚拟中断处理流程

中断控制器是模拟的控制器，中断处理是先由主机处理，再转发给vCPU处理

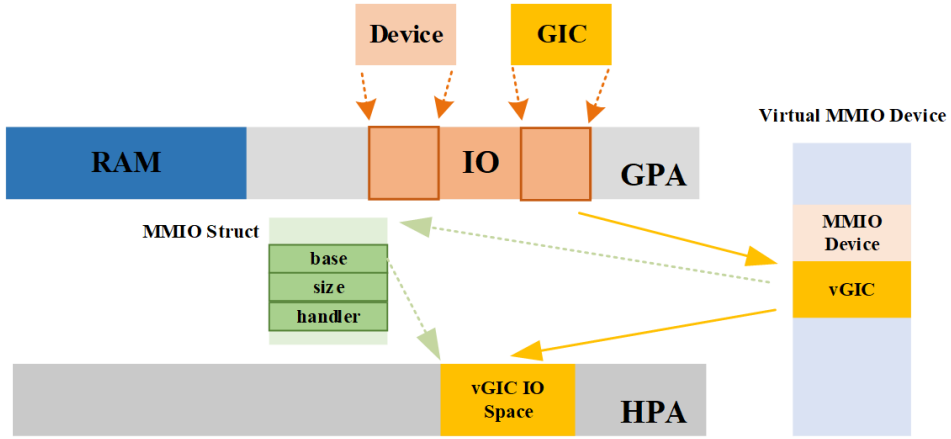
虚拟外设模块设计与实现

虚拟外设映射

➢ **MMIO虚拟设备映射框架：**

- 当虚拟机访问一个虚拟的I/O设备时，可以通过内存中提前配置好的Virtual MMIO Device来实现设备访问。

- MMIO struct保存虚拟设备信息，其中主要通过回调函数进行自定义处理。如图中虚拟机进行GIC设备的操作时，会触发异常并调用handler回调函数，以访问到Virtual MMIO Device，实现设备的虚拟化。



虚拟外设MMIO框架

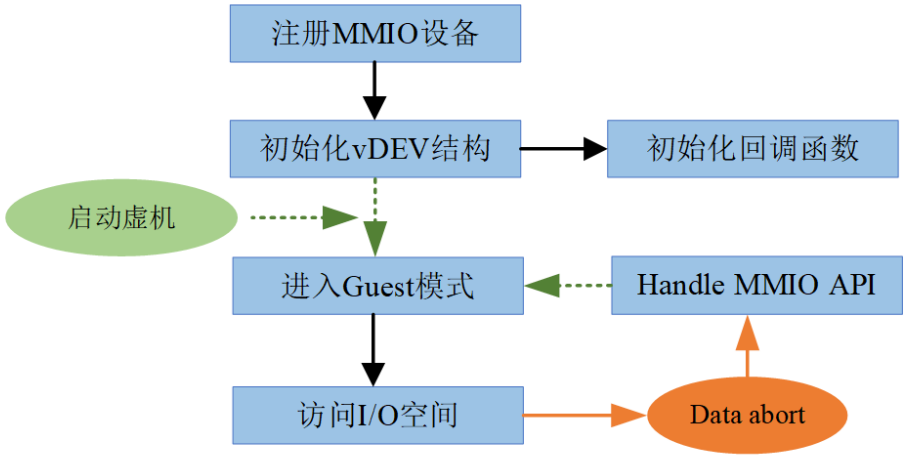
设备虚拟化的2个关键技术：vDec设备的构造，vCPU如何访问vDec

虚拟外设模块设计

虚拟外设模块设计

➢ **此模块主要分为如下两个部分：**

- 在VM创建过程中，需要创建一些Virtual MMIO（Memory-mapped I/O,将设备地址映射到物理内存地址空间进行访问）设备，为VM访问相应设备构造好地址空间和处理机制）
- 在VM执行过程中，当访问到I/O空间时，将触发地址访问错误（vCPU无法直接访问Virtual MMIO device地址），Hypervisor调用回调函数来模拟device访问操作。而这个Virtual MMIO device地址通过MMIO映射到物理内存空间，从而vCPU可以通过Hypervisor访问到Virtual MMIO Device



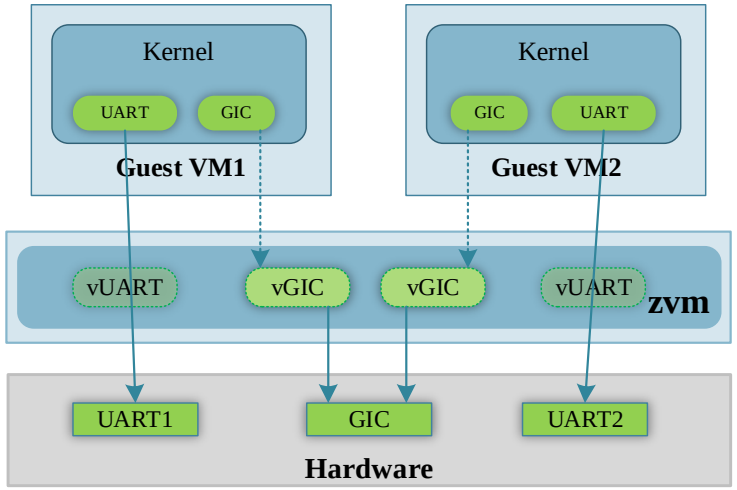
虚拟外设执行流图

设备是模拟的设备，设备访问是代理式访问

虚拟外设模块设计

完全设备虚拟化

- **完全虚拟化设备**：由软件模拟，兼容性好，开销大，性能差。用于**不可单独分配设备**（例如中断控制器GIC）。
- 为每个VM提供了一个完全虚拟化的GIC设备，并为其在内存中分配一段地址，模拟GIC的IO地址空间，并存储当前vGIC的配置信息；当执行VM时，将配置信息通过Hypervisor控制写入物理GIC地址当中或者完全通过软件模拟GIC访问操作，以实现GIC设备的虚拟化



中断设备是模拟的设备，中断设备访问是代理式访问

虚拟外设模块设计

设备直通

- **设备直通**：使用原有的硬件驱动，不增加新驱动，性能和主机上直接使用该物理硬件非常接近。用于**可单独分配设备**（调试串口等独占外设）
- 如果在VM访问前， 系统提前建立了二阶段地址映射的话，vCPU就可以直接访问真实设备
- 直通如果涉及DMA操作的话就需要SMMU（将虚拟机物理地址空间内的GPA翻译成HPA）支持，因为DMA（外设可以通过DMA，将数据批量传输到物理内存）必须使用真实的物理地址，使用虚拟地址会报错。
- 直通在VM初始化过程中分配给指定的VM，实现VM对该设备的直接使用，而Hypervisor在此过程中只需要记录设备分配给了哪个VM，不需要进行具体设备功能的模拟，减小了系统的开销。

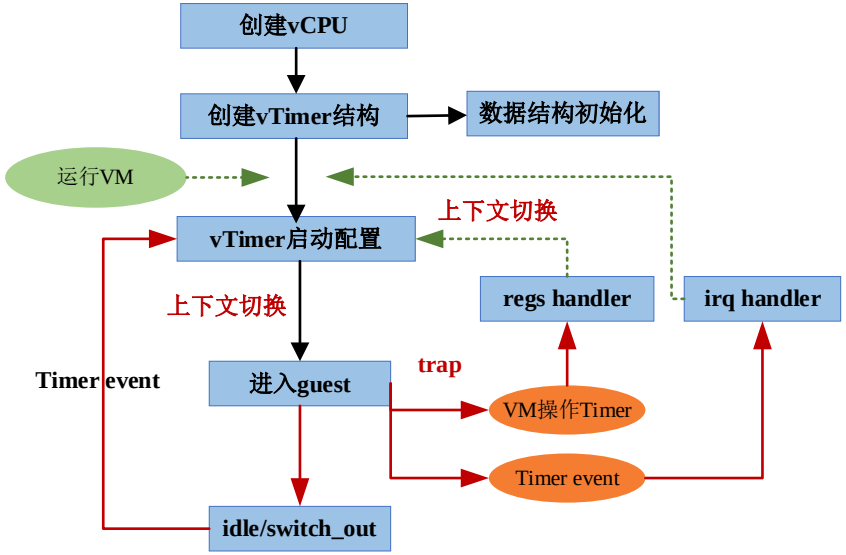
设备是真实的设备，设备访问是直接式访问

虚拟时钟模块设计

vTimer模块简介

➢ **时钟虚拟化：**

- vTimer用于给vCPU提供**虚拟定时器**，满足VM中需要定时器的服务（比如VM需要在1秒后使用某个设备）。
- 每次vTimer发生计数器中断后需要Hypervisor处理,一样是硬件中断，只不过它用的是专门的面向虚拟时钟的寄存器，每个CPU定义了一组虚拟时钟寄存器，它们单独计数并在预定的时间过后抛出中断，并由主机转发至VM



vTimer执行流

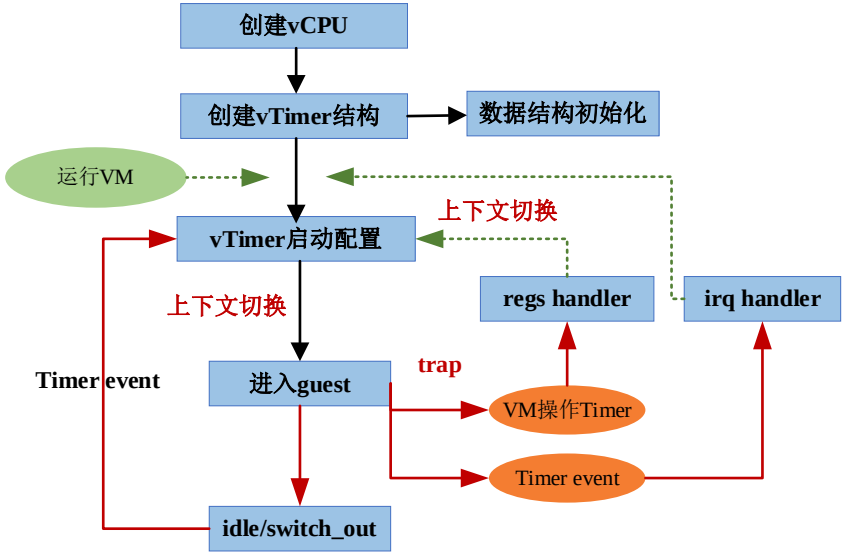
时钟虚拟化关键技术： 1） vTimer的构造； 2）vTimer定时事件如何被vCPU感知到

虚拟时钟模块设计

vTimer模块执行逻辑

➢ **vTimer：**

- vCPU初始化时初始化vTimer并配置中断服务程序**ISR的异常处理函数**及**timeout事件处理函数**。
- 通过配置Compare值，实现**vTimer中断**，以支持调度功能。
- vCPU空闲或者退出处理器时，使用timeout事件处理函数注入中断并**调度vCPU**。



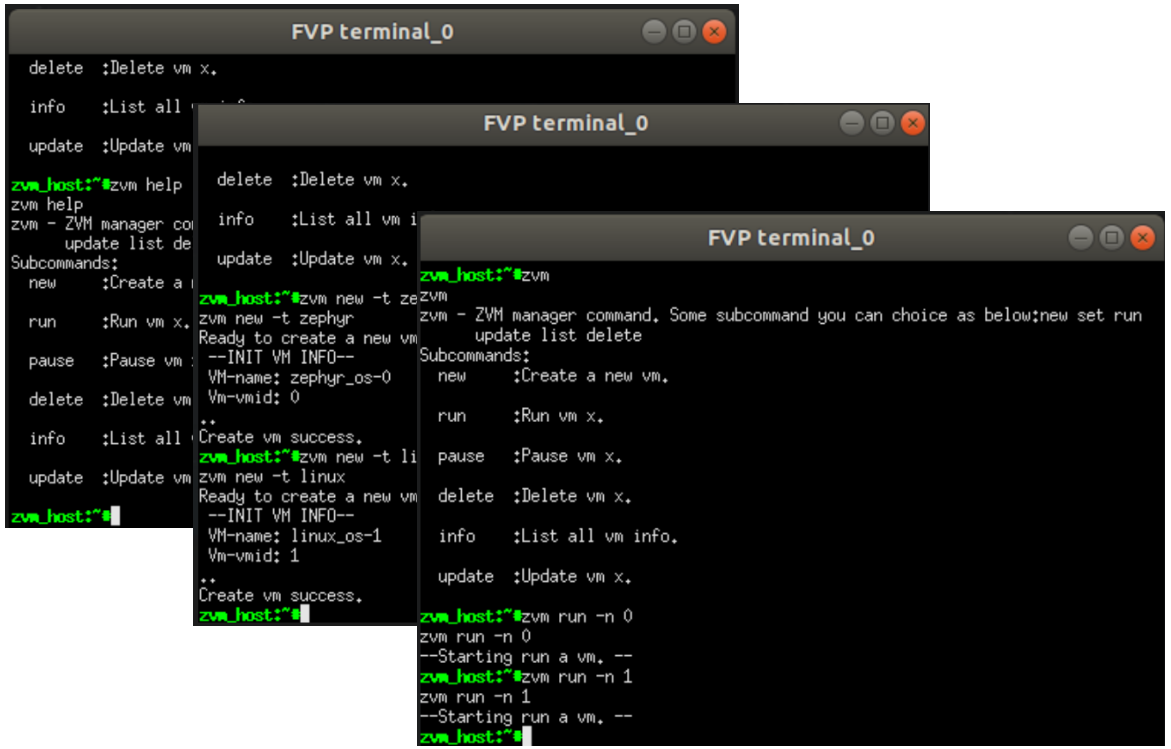
vTimer执行流

vTimer定时事件是一个硬件中断，触发事件之后就走中断流程

具体实现

虚拟机环境下的基本操作

- **演示平台**
- FVP Cortex-A55 Platform
- **支持外设**
- 中断控制器（GICv3）
- 调试串口（PL011）
- **主机支持命令**
- new, run, pause, delete, info
- **虚拟机支持**
- 调试串口打印系统或内核信息



虚拟机操作演示示例

开源共享

ZVM开源

sig-zephyr 2023年1月例会 (2023-02-01)

258 0 2023-02-06 17:50:23 未经作者授权，禁止转载

sig-zephyr 2023年1月例会 (2023-02-01)

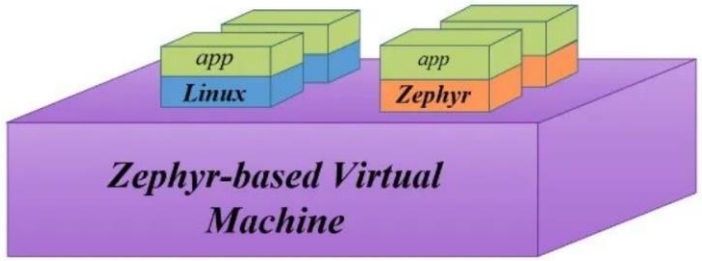
Sig-zephyr 项目组技术分享

openEuler开源新项目，嵌入式实时虚拟机ZVM介绍

原创 湖大嵌入式实验室 openEuler 2023-03-23 18:00 发表于香港

嵌入式实时虚拟机ZVM（Zephyr-based Virtual Machine）是由湖南大学嵌入式与网络计算湖南省重点实验室（以下简称"湖大嵌入式实验室"）主任谢国琪教授主导设计并开发的虚拟化软件。该软件基于实时操作系统（RTOS）Zephyr开发，可同时启动Linux与Zephyr 2个Guest OS，从而在同一硬件平台上实现混合内核部署。

2023年2月，湖大嵌入式实验室正式将该项目开源至openEuler社区。

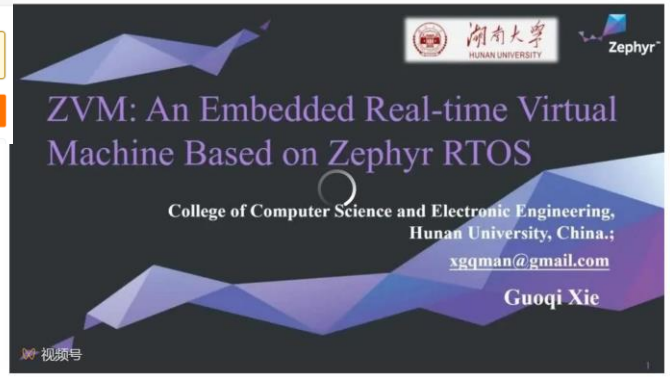


openEuler开源仓库：

https://gitee.com/openeuler/zvm

全球嵌入式开源峰会2023：嵌入式实时虚拟机ZVM视频分享

原创 嵌入式计算湖南省重点实验室 2023-07-04 00:00 发表于湖南



ZVM实时性优化

Zephyr & hypervisor & Real Time

➢ **复杂嵌入式系统实时性需求**

- 需要支持一些通信协议（如CAN等协议）
- 需要支持一些外设（各类传感器）
- 需要支持多任务运行（裸机系统不再适用）
- 需要支持系统安全隔离（基于虚拟化的应用）

实时系统中部署系统虚拟化

➢ **基于Zephyr实时系统虚拟化优势**

- 开发者专注实时应用开发（不需考虑底层）
- 同时部署实时/非实时系统任务（多系统支持）
- 资源的隔离和共享（系统资源虚拟化）
- 使用Zephyr实时性支持（依赖主机特性）

用Zephyr RTOS实时特性

ZVM实时性优化

Zephyr & hypervisor & Real Time

➢ Zephyr实时性支持（线程为调度的基本单位）

- 调度时机：线程状态改变、线程间通信信号、主动的线程上下文切换以及中断的响应都会引发调度
- 某一时刻线程调度的优先级是确定的
- 优先级：支持协作和抢占式调度、支持多级抢占式优先级调度，可以在运行时调整线程的优先级（便于支持提升优先级、优先级置顶等防止优先级倒置的操作）
- 支持多种调度队列，支持SMP系统，支持线程亲和性配置
- 支持中断嵌套（多级中断处理）

➢ ZVM系统实时性特性

- 支持抢占优先级调度，vCPU线程参与主机调度过程
- 复用主机线程切换逻辑，减少线程切换开销
- 支持中断快速路径，减少中断虚拟化处理开销
- 使用Linux和RTOS线程标记，将其部署在不同的物理CPU并分配不同的优先级

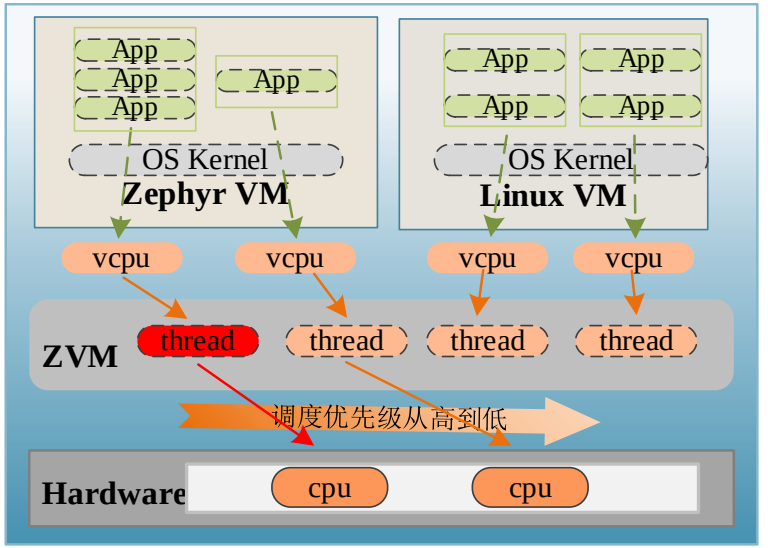
设计目标：降低虚拟化系统时延，依靠Zephyr Host实时调度策略保证系统实时性。

ZVM实时性优化

Zephyr & hypervisor & Real Time

➢ **Zephyr端多任务情况（单个多应用Zephyr VM情况）**

- 使用单个Zephyr VM中跑多个Zephyr应用，并将应用负载反馈给主机线程(**正在进行开发**)。
  - 在单个Zephyr VM中运行**多个应用程序**;
  - 系统调度优先级是**主机线程 > Zephyr vCPU > Linux vCPU**;
  - 现有问题：解决如何保证单个Zephyr VM中应用的实时性问题，即带vCPU负载感知的主机调度策略（**正在调研方案**）



虚拟内存方案实现与优化

ZVM设备内存管理设计

➢ dtb解析功能

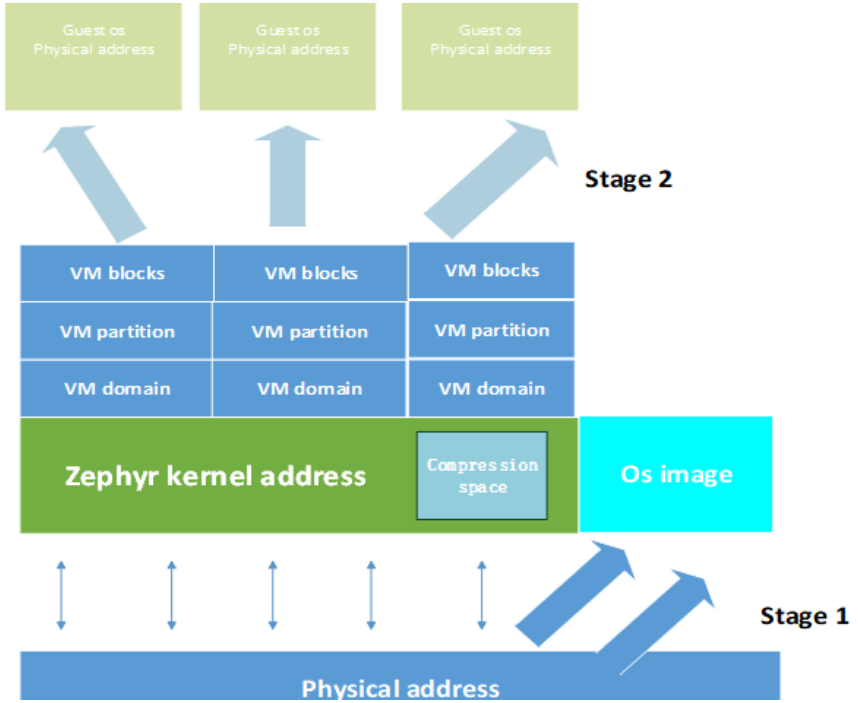
- zvm实现dtb解析功能，可以根据VM传入的dtb文件解析设备树，并分配设备地址空间。

➢ 内核镜像重定位

- 编译期将内核镜像加载到指定内存空间中。

➢ 支持内存管理

- 支持动态内存分配，支持为虚拟机动态分配内存，进一步支持虚拟内存的缺页处理机制。



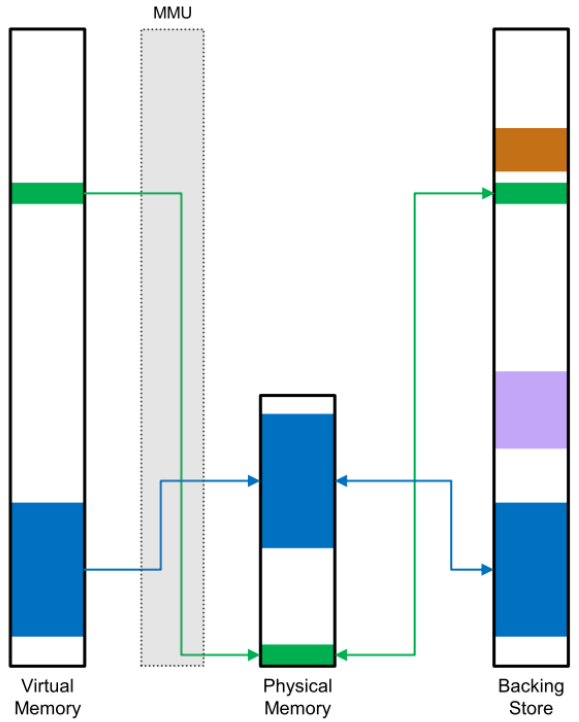
内存多级管理架构

虚拟内存方案实现与优化

Demand paging在ZVM部署

ZVM所需要虚拟地址大于物理地址时，需要构建页swap机制：

- **应用场景：**
  - 需要分配给虚拟机较大的虚拟物理地址空间地址空间
  - 内存资源受限设备上运行较大Linux等系统
- **面向问题：**
  - 设备有后备存储资源（磁盘），并被Zephyr支持
  - 设备必须支持MMU转换机制
  - 如何解决SWAP过程中实时性的问题



Demand paging

虚拟设备框架方案设计

ZVM virtIO设计

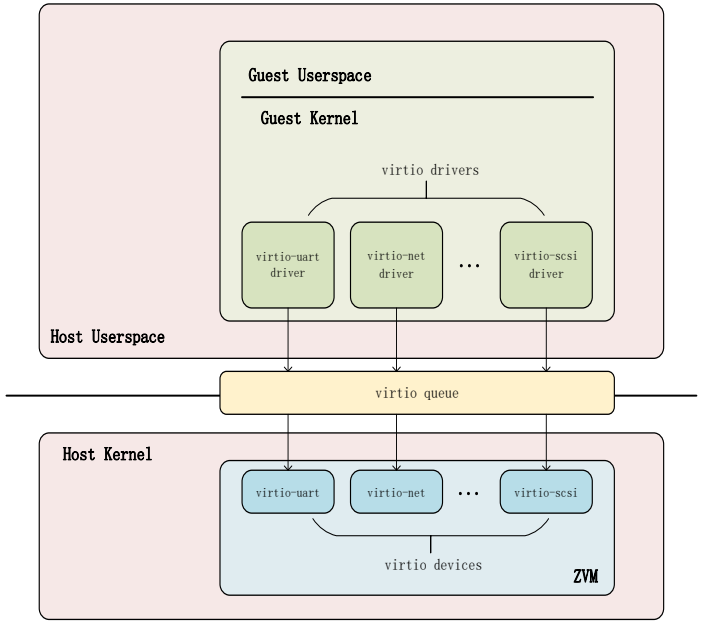
➢ 支持设备模型（减轻ZVM驱动移植难度）

实现方案：

- 以Linux 2.6.25-rc6（第一次合并virtI/O的Linux内核版本）中的virtI/O实现为参考对象
- 开发路线：虚拟队列virtio-queue→虚拟驱动程序virtio-driver→虚拟设备virtio-device

进度：

- 进行virtio-queue的设计与实现，zephyr和Linux中的设备管理模块有较大的差异，需要重构设备管理模块的许多数据结构



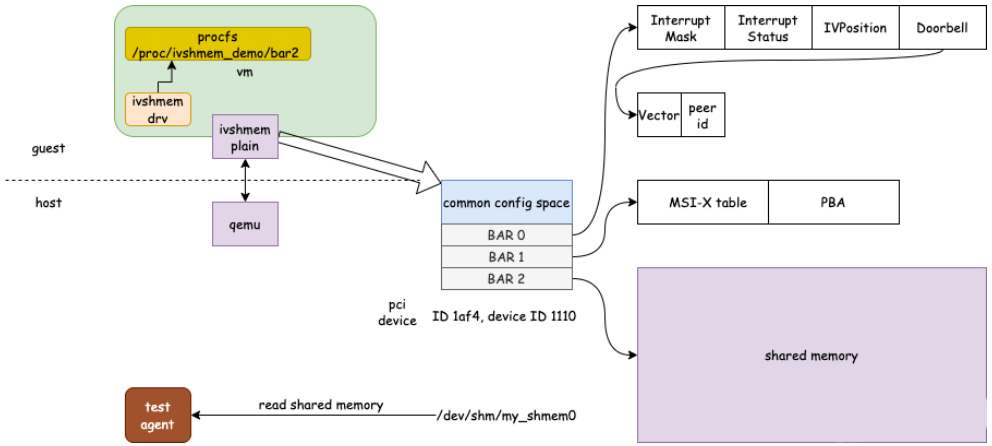
virtIO拟实现框架

实时通信方案测试

Linux下Zephyr实时通信测试

提供ZVM虚拟机间实时通信方案

- **在Ubuntu下配置开发编译环境**
  - 在Linux环境下作为同时开启多个zephyr，作为应用程序运行。
  - 使用QEMU模拟运行Zephyr 。
- **虚拟机间共享内存 (ivshmem)**
  - 共享运行不同客户机的多个QEMU进程以及与宿主机之间的内存区域
  - 将内存映射成guest内的pci设备，在非中断模式下直接把虚拟pci设备当做一个共享内存进行操作



Qemu Ivshmem参考架构

THANKS FOR ALL

嵌入式与网络计算湖南省重点实验室

http://esnl.hnu.edu.cn/