ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences

openEuler Embedded支持RISC-V体系架构

中国科学院软件研究所

于佳耕

汇报提纲

1. openEuler Embedded 介绍

2. openEuler Embedded 适配 VisionFive2

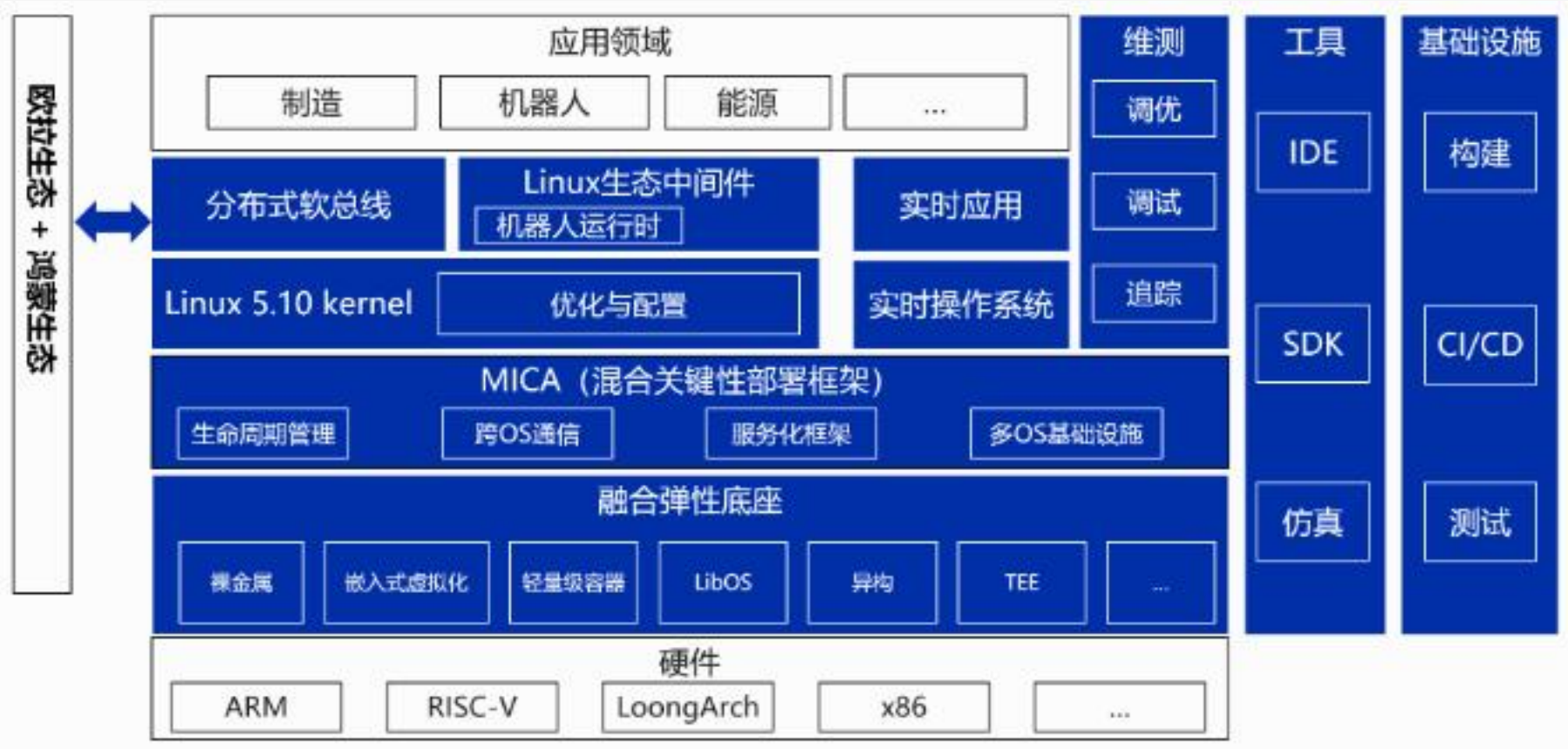
3. RISC-V分布式软总线

4. Musl libc的RVV优化

5. RISC-V 板卡对接 openEuler Embedded 流程与建议

openEuler Embedded 介绍

openEuler Embedded 是基于 openEuler 社区面向嵌入式场景的 Linux 版本，旨在成为一个高质量的嵌入式 Linux。其在内核版本、软件包版本等代码层面会与 openEuler 其他场景的 Linux 保持一致，共同演进，不同之处在于针对嵌入场景的内核配置、软件包的组合与配置、代码特性补丁的不同。



openEuler Embedded总体架构

openEuler Embedded 介绍

openEuler Embedded 当前采用 Yocto 来构建嵌入式 Linux，实现了与 openEuler 其他版本代码同源。yocto-meta-openeuler 是用于构建 openEuler Embedded 的一系列构建配方的集合，也就是配方层的集合，以及包含相关构建工具和 openEuler Embedded 的开发使用文档。

yocto-meta-openeuler 目录架构：

- scripts：一系列辅助工具，用于帮助构建环境，如下载代码仓、创建构建环境等等
- meta-openeuler：构建openEuler Embedded所创建的Yocto层，包含相应的配置、构建配方等等
- bsp：openEuler Embedded的BSP(Board Support Package)抽象层，包含当前openEuler Embedded所支持的硬件BSP, 如QEMU、树莓派4B等等
- RTOS：openEuler Embeddd的RTOS(Real-Time Operating System)抽象层，主要针对Linux和RTOS混合关键部署的场景，当前支持RT-Thread和Zephyr
- docs：openEuler Embedded使用和开发文档， CI会自动构建文档，并发布于如下地址：openEuler Embedded开发使用文档

汇报提纲

1. openEuler Embedded 介绍

2. openEuler Embedded 适配 VisionFive2

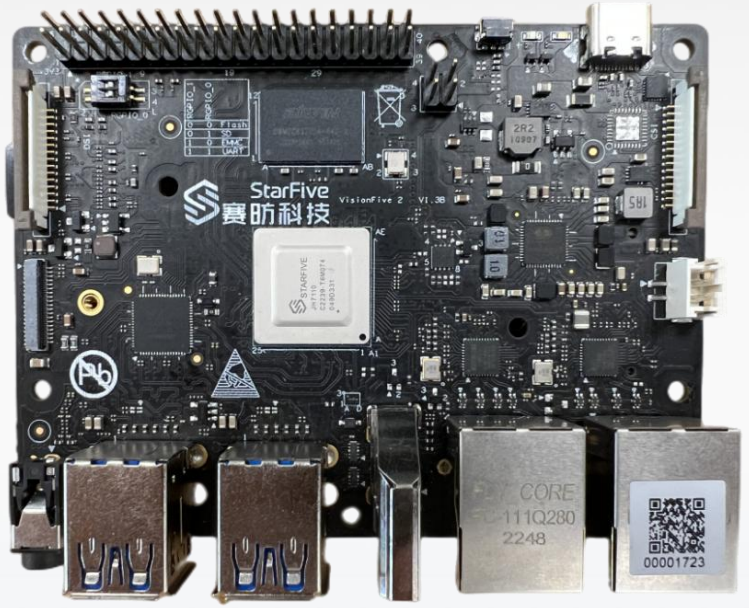
3. RISC-V分布式软总线

4. Musl libc的RVV优化

5. RISC-V 板卡对接 openEuler Embedded 流程与建议

openEuler Embedded 适配 VisionFive2

- 提供 2/4/8 GB LPDDR4 RAM 选项；
- 数据传输速率最高可达 2800Mbps；
- 外设 I/O 接口丰富，包括 M.2 接口、eMMC 插座、USB 3.0 接口、40-pin GPIO header、千兆以太网接口、TF 卡插槽等；
- 配备赛昉科技推出的四核 64 位 RV64GC ISA 的 SoC JH7110，具有更加强大的 GPU 处理能力和多媒体支持能力；
- 搭载 64 位高性能四核 RISC-V CPU，工作频率最高可达 1.5 GHz；
- 板载的 Imagination BXE-4-32 GPU，支持 OpenCL 3.0，OpenGL ES 3.2和 Vulkan 1.2;
- 支持 Linux、Debian、OpenEuler、聚元PolyOS 和 OpenHarmony 操作系统。



VisionFive 2：集成3D GPU的 RISC-V SBC

openEuler Embedded 适配 VisionFive2

- VisionFive 2 开发板镜像可基于最新的 yocto-meta-openeuler 进行构建和试用，采用欧拉开源固件和 5.10 版本内核分支。
- 用户可以使用 oebuild 快速构建 openEuler Embedded，VisionFive2 构建时只需选择对应硬件平台即可，当前只支持在 X86 64 位的 Linux 环境下构建 openEuler Embedded。

meta-visionfive2

meta-visionfive2 访问地址

openEuler embedded 编译构建架构

oebuild 是 openEuler Embedded 孵化的一个开源项目，是为了辅助开发 openEuler Embedded 项目而衍生的辅助开发工具。

1、oebuild init <directory>

该操作会初始化 oebuild 的目录，<directory> 表示要初始化目录的名称

2、oebuild update

①.pull 相关的运行容器镜像

②.从 gitee 上下载 yocto-meta-openeuler 仓代码，如果本地没有 openeuler 相关容器，则在这一步执行会比较漫长，请耐心等待。

3、oebuild generate -p visionfive2

产生 visionfive2 编译配置文件——compile.yaml

4、oebuild bitbake openeuler-image

构建 openeuler-image 镜像

汇报提纲

1. openEuler Embedded 介绍

2. openEuler Embedded 适配 VisionFive2

3. RISC-V分布式软总线

4. Musl libc的RVV优化

5. RISC-V 板卡对接 openEuler Embedded 流程与建议

定制分布式软总线功能 meta-layer

分布式软总线技术是基于华为多年的通信技术积累，参考计算机硬件总线，在 1+8+N 设备间搭建一条“无形”的总线，具备自发现、自组网、高带宽、低时延的特点。当前 openEuler、OpenHarmony 社区均支持分布式软总线。

1+8+N概念：

1：手机

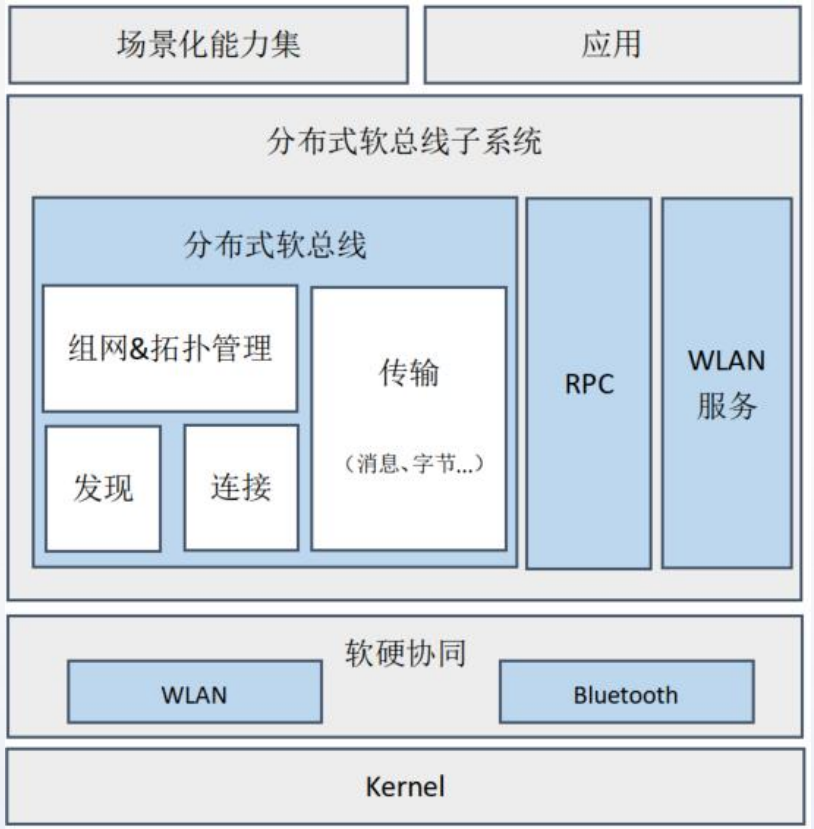
8：车机、音箱、耳机、手表/手环、平板、大屏、PC、AR/VR

N：其他IOT设备

定制分布式软总线功能 meta-layer

分布式软总线子系统旨在为 OpenHarmony 系统提供的通信相关的能力。

- **WLAN 服务**：为用户提供 WLAN 基础功能、 P2P（peer-to-peer）功能和 WLAN 消息通知的相应服务，让应用可以通过 WLAN 和其他设备互联互通。
- **蓝牙服务**：为应用提供传统蓝牙以及低功耗蓝牙相关功能和服务。
- **软总线**：为应用和系统提供近场设备间分布式通信的能力，提供不区分通信方式的设备发现，连接，组网和传输功能。
- **进程间通信**：提供不区分设备内或设备间的进程间通信能力。



定制分布式软总线功能 meta-layer

- dsoftbus_standard
  提供了对软总线系统功能基本的支持

- embedded-ipc
  支持在嵌入式设备中进行低开销和高性能的IPC通信服务

- yocto-embedded-tools
  提供了编译软总线系统所必须的工具

- dsoftbus_1.0.bb
  软总线配方文件，安装软总线依赖的安装包、三方库和软总线应用程序

```
.
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README.md
└── recipes-dsoftbus
    └── dsoftbus
        ├── dsoftbus_1.0.bb
        └── files
            ├── dsoftbus_standard
            ├── embedded-ipc
            ├── libboundscheck
            └── yocto-embedded-tools
```

meta-dsoftbus

meta-dsoftbus 访问地址

定制分布式软总线功能 meta-layer

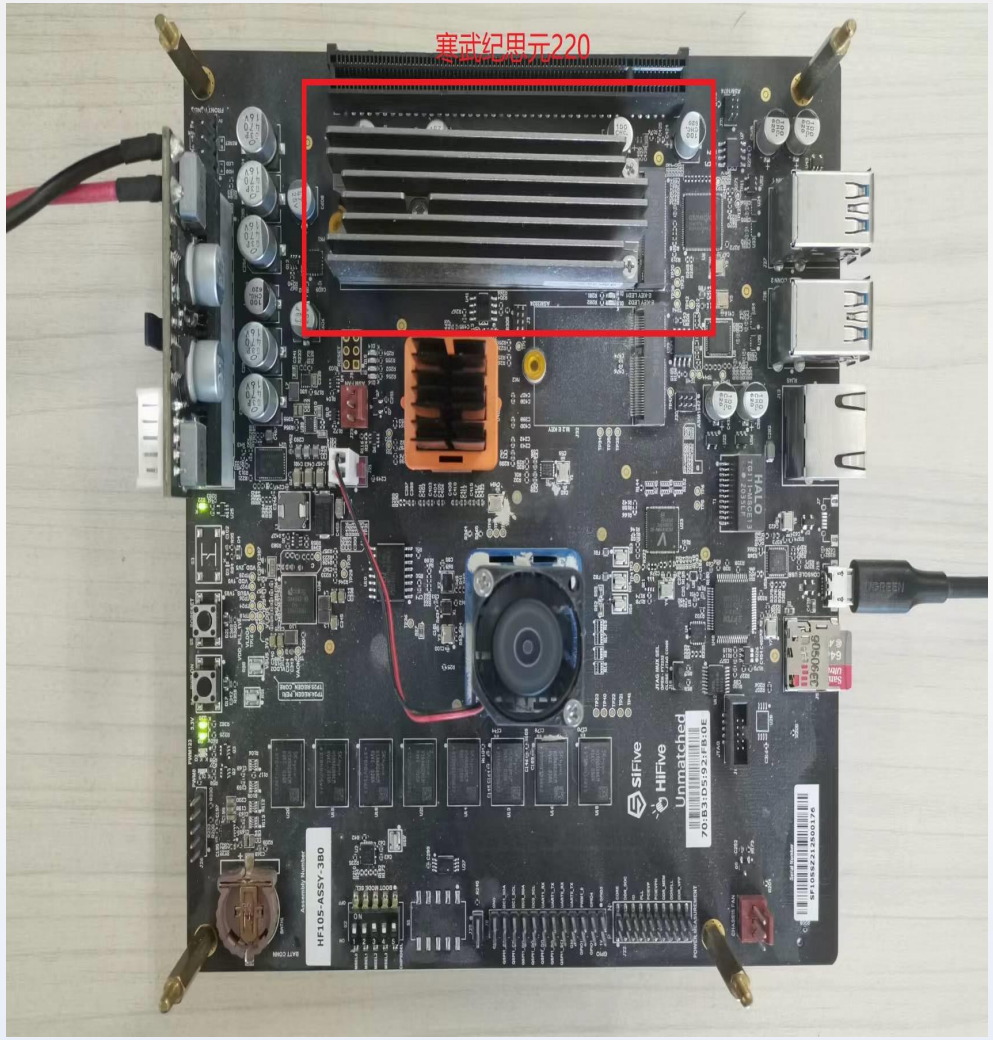

```
libcoap.z.so                          libnstackx_util.open.z.so
libdeviceauth_sdk.z.so                libsec_shared.z.so
libhilog.z.so                         libsoftbus_adapter.z.so
libhuks_engine_core_standard.z.so     libsoftbus_client.z.so
libhukssdk.z.so                       libsoftbus_server.z.so
libipc_core.z.so                      libsoftbus_utils.z.so
libmbedtls.z.so                       libsyspara.z.so
libnstackx_ctrl.z.so                  libutils.z.so
```

- libsoftbus_server.z.so：支持软总线的基础server服务；
- libsoftbus_client.z.so：支持软总线的基础client通信服务；
- libdeviceauth_sdk.z.so：提供可信设备的认证服务；
- libhilog.z.so：提供软总线系统的日志输出服务，根据IOT设备的资源丰富程度，提供不同级别的日志输出；
- libsyspara.z.so：提供服务读取系统中的/etc/SN文件接口，并生成局域网内的设备唯一标识符。

构建基于 RISC-V 的 Linux 系统设备对接 HarmonyOS

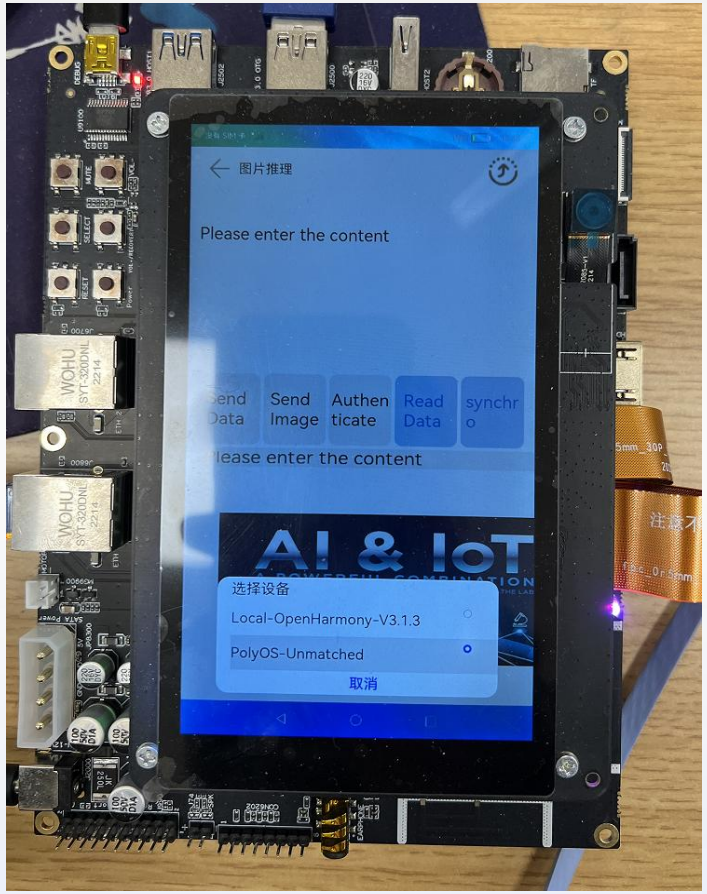
- 寒武纪 Cntoolkit 移植到 RISC-V
- 寒武纪 CNStream 依赖的第三方库移植到 RISC-V
- 寒武纪 CNStream 移植到 RISC-V
- 寒武纪 软件栈 与 PolyOS AIoT 对接

在终端智能推理性能达到业界领先



演示软总线应用 Demo

OpenHarmony & RK3568



设备认证

设备发现与连接

图片传输

推理结果回送

Sifive Unmatched & PolyOS & MLU220

汇报提纲

1. openEuler Embedded 介绍

2. openEuler Embedded 适配 VisionFive2

3. RISC-V分布式软总线

4. Musl libc的RVV优化

5. RISC-V 板卡对接 openEuler Embedded 流程与建议

关于musl libc

中国科学院软件研究所 Institute of Software Chinese Academy of Sciences

musl-1.2.4

- first released in 2011
- 基于Linux系统调用API构建
- 遵循ISO C和POSIX标准
- MIT license

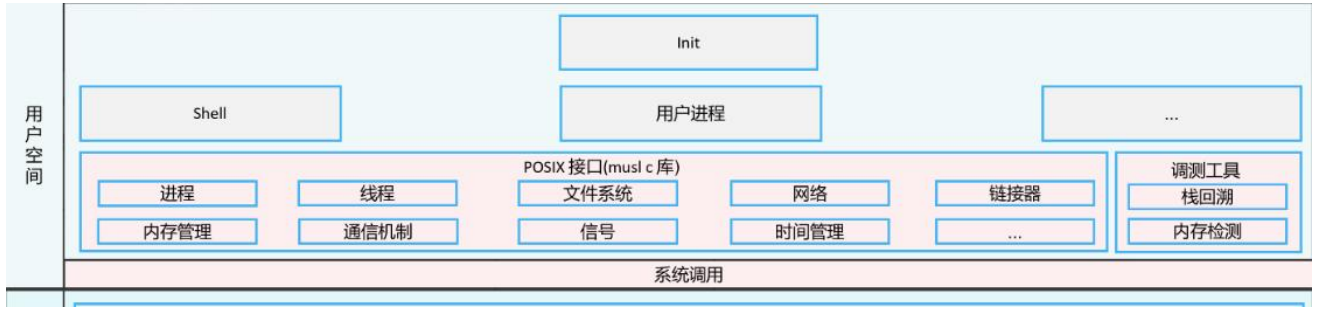
Bloat comparsion

Bloat comparison	musl	uClibc	dietlibc	glibc
Complete .a set	426k	500k	120k	2.0M †
Complete .so set	527k	560k	185k	7.9M †
Smallest static C program	1.8k	5k	0.2k	662k
Static hello (using printf)	13k	70k	6k	662k
Dynamic overhead (min. dirty)	20k	40k	40k	48k
Static overhead (min. dirty)	8k	12k	8k	28k
Static stdio overhead (min. dirty)	8k	24k	16k	36k
Configurable featureset	no	yes	minimal	minimal

Performance comparsion

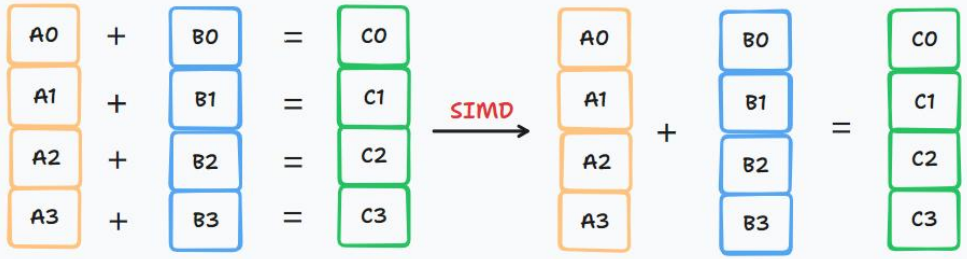
Performance comparison	musl	uClibc	dietlibc	glibc
Tiny allocation & free	0.005	0.004	0.013	0.002
Big allocation & free	0.027	0.018	0.023	0.016
Allocation contention, local	0.048	0.134	0.393	0.041
Allocation contention, shared	0.050	0.132	0.394	0.062
Zero-fill (memset)	0.023	0.048	0.055	0.012
String length (strlen)	0.081	0.098	0.161	0.048
Byte search (strchr)	0.142	0.243	0.198	0.028
Substring (strstr)	0.057	1.273	1.030	0.088
Thread creation/joining	0.248	0.126	45.761	0.142
Mutex lock/unlock	0.042	0.055	0.785	0.046
UTF-8 decode buffered	0.073	0.140	0.257	0.351
UTF-8 decode byte-by-byte	0.153	0.395	0.236	0.563
Stdio putc/getc	0.270	0.808	7.791	0.497
Stdio putc/getc unlocked	0.200	0.282	0.269	0.144
Regex compile	0.058	0.041	0.014	0.039
Regex search (a{25}b)	0.188	0.188	0.967	0.137
Self-exec (static linked)	234µs	245µs	272µs	457µs
Self-exec (dynamic linked)	446µs	590µs	675µs	864µs

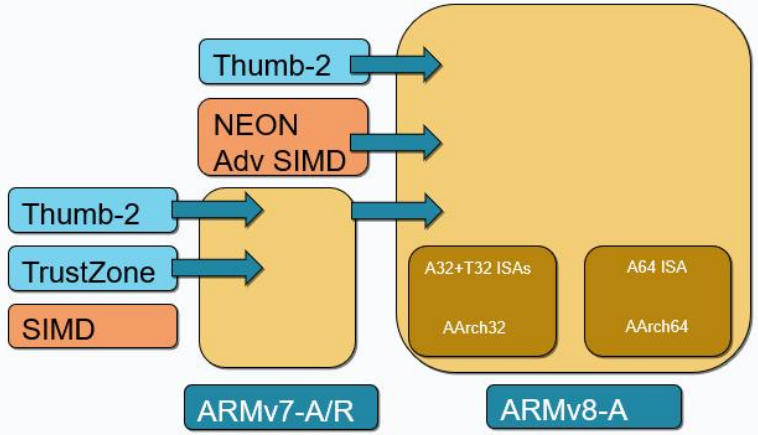
OpenHarmony LiteOS-A

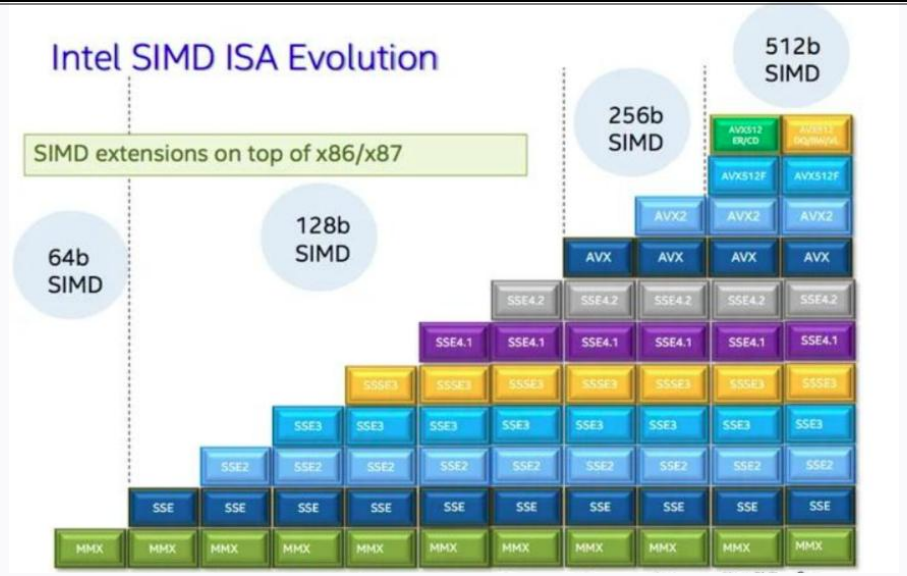


关于RVV

• SIMD







riscv-v-spec

Working draft of the proposed RISC-V V vector extension.

Version 1.0 has been frozen and at this time is undergoing public review. Version 1.0 is considered stable enough to begin developing toolchains, functional simulators, and implementations, including in upstream software projects, and is not expected to have incompatible changes except if serious issues are discovered during ratification. Once ratified, the spec will be given version 2.0.

The *previous* stable releases are v1.0-rc2, v1.0-rc1, v0.10, v0.9, and v0.8. Note, these previous releases were for experimental development purposes only and are not standard versions suitable for production use. Significant incompatible changes were made from these earlier versions prior to freezing.

The top level file is v-spec.adoc

Simply clicking on the file in the github repo viewer will render a usable version as markdown.

For a better rendering, use the documentation build process described below.

Additional Resources

- The Spike simulator supports v1.0.
- The RISC-V Proxy Kernel (to be used with e.g. Spike) supports v1.0 binaries.
- The Binutils port for v0.8
- The GNU toolchain port for v0.8
- riscvOVPsim is a free RISC-V reference simulator that has support for v0.9, v0.8 and v0.7.1 (simulator is under a proprietary license, models are open source)

优化目标

Glibc	aarch64	x86_64	riscv
memset	memset.S memset_falkor.S memset_kunpeng.S memset_a64fx.S	memset-sse2-unaligned-erms.S memset-avx2-unaligned-erms.S memset-avx512-unaligned-erms.S	memset.c
memcpy	memcpy.S memcpy_sve.S memcpy_falkor.S memcpy_a64fx.S	memmove-sse2-unaligned-erms.S memmove-avx-unaligned-erms.S memmove-avx512-unaligned-erms.S	memcpy.c
memmove	memcpy.S memcpy_sve.S memcpy_falkor.S memcpy_a64fx.S	memmove-sse2-unaligned-erms.S memmove-avx-unaligned-erms.S memmove-avx512-unaligned-erms.S	memmove.c
strlen	strlen.S strlen_asimd.S	strlen-sse2.S strlen-avx2.S strlen-evex.S	strlen.c
…	…	…	…

musl	aarch64	x86_64	riscv
memcpy	memcpy.s	memcpy.s	memcpy.c
memset	memset.s	memset.s	memset.c
memmove	memmove.c	memmove.s	memmove.c

“靶子”

- 内存操作函数
- 字符串操作函数

RVV兼容方案

- RISC-V指令集采用模块化设计
  - 必要的RV32I只有47条指令
  - 其余指令可选扩展

硬件不同，指令集支持情况不同。需考虑兼容性问题!

读取misa寄存器，判断指令集支持情况

```
438     #ifdef CONFIG_RISCV_ISA_V
439             csrr    t0, CSR_MISA
440             li      t1, COMPAT_HWCAP_ISA_V
441             and     t0, t0, t1
442             beqz    t0, .Lreset_regs_done_vector
443
444             /*
445              * Clear vector registers and reset vcsr
446              * VLMAX has a defined value, VLEN is a constant,
447              * and this form of vsetvli is defined to set vl to VLMAX.
448              */
449             li      t1, SR_VS
450             csrs    CSR_STATUS, t1
451             csrs    CSR_VCSR, x0
452             vsetvli t1, x0, e8, m8, ta, ma
453             vmv.v.i v0, 0
454             vmv.v.i v8, 0
455             vmv.v.i v16, 0
456             vmv.v.i v24, 0
457             /* note that the caller must clear SR_VS */
458     .Lreset_regs_done_vector:
459     #endif /* CONFIG_RISCV_ISA_V */
460             ret
```

misa寄存器

The `misa` CSR is a **WARL** read-write register reporting the ISA supported by the hart. This register must be readable in any implementation, but a value of zero can be returned to indicate the `misa` register has not been implemented, requiring that CPU capabilities be determined through a separate non-standard mechanism.

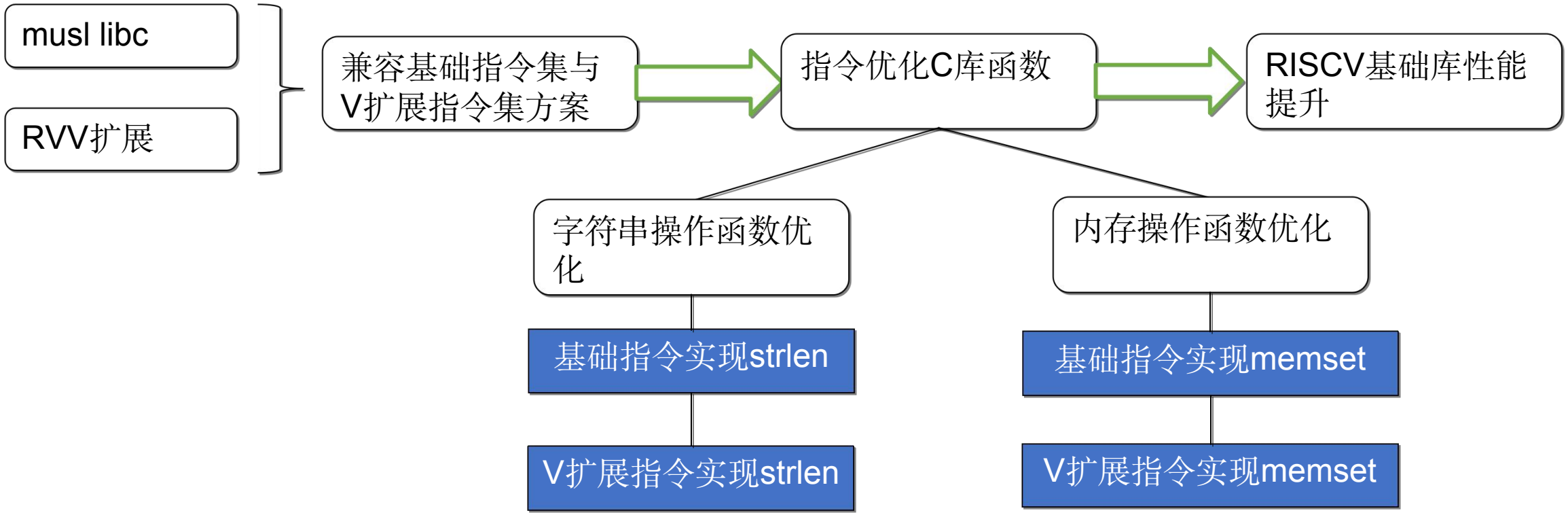
MXLEN-1 MXLEN-2	MXLEN-3 26	25 0
MXL[1:0] (**WARL**)	0 (**WARL**)	Extensions[25:0] (**WARL**)
2	MXLEN-28	26

Machine ISA register (misa).

- hwcap
  - 硬件探测
  - 运行时开销

来源于：linux/arch/riscv/kernel/head.S

优化目标


musl libc
RVV扩展
兼容基础指令集与V扩展指令集方案
指令优化C库函数
RISCV基础库性能提升
字符串操作函数优化
内存操作函数优化
基础指令实现strlen
V扩展指令实现strlen
基础指令实现memset
V扩展指令实现memset

测试环境

软件	网址
gem5	https://github.com/plctlab/plct-gem5
gcc	https://github.com/riscv-collab/riscv-gnu-toolchain.git
ARM测试集	https://github.com/ARM-software/optimized-routines/tree/master/string/bench

strlen测试

表1 small aligned strlen性能对比(单位：bytes/ns)

数据量	musl C语言实现	基础指令汇编实现	RVV汇编实现
1B	0.04	0.04	0.03
2B	0.08	0.08	0.07
4B	0.13	0.14	0.14
8B	0.21	0.24	0.27
16B	0.38	0.43	0.55
32B	0.62	0.73	1.10
64B	0.92	1.10	2.20
平均	0.34	0.39	0.62

表2 small unaligned strlen性能对比(单位：bytes/ns)

数据量	musl C语言实现	基础指令汇编实现	RVV汇编实现
1B	0.05	0.05	0.03
2B	0.09	0.09	0.07
4B	0.14	0.14	0.13
8B	0.17	0.17	0.26
16B	0.31	0.32	0.52
32B	0.53	0.57	1.05
64B	0.81	0.91	2.09
平均	0.30	0.32	0.59

表3 medium strlen性能对比(单位：bytes/ns)

数据量	musl C语言实现	基础指令汇编实现	RVV汇编实现
128B	1.23	1.51	4.46
256B	1.45	1.82	9.28
512B	1.60	2.03	12.46
1K	1.68	2.15	15.04
2K	1.73	2.22	16.78
4K	1.75	2.25	17.80
平均	1.57	2.00	12.64

实验结果分析：

1.基础指令实现的strlen与musl C语言实现性能相当；

2. RVV实现的strlen相比于C实现，small aligned测试性能平均提升**83%**，small unaligned测试性能平均提升**98%**，medium测试性能平均提升**703%**。

memset测试

表1 random memset性能对比(单位：bytes/ns)

数据量	musl C语言实现	基础指令汇编实现	RVV汇编实现
32K	0.67	0.81	1.25
64K	0.67	0.80	1.24
128K	0.68	0.81	1.25
256K	0.67	0.80	1.24
512K	0.67	0.80	1.24
1024K	0.67	0.80	1.23
平均	0.67	0.80	1.24

表2 medium memset性能对比(单位：bytes/ns)

数据量	musl C语言实现	基础指令汇编实现	RVV汇编实现
8B	0.39	0.36	0.53
16B	0.49	0.54	1.06
32B	0.75	1.05	2.12
64B	1.32	1.96	3.97
128B	2.26	3.05	7.06
256B	3.53	6.72	11.57
512B	4.89	9.29	17.59
平均	1.95	3.28	6.27

表3 large memset性能对比(单位：bytes/ns)

数据量	musl C语言实现	基础指令汇编实现	RVV汇编实现
1K	6.02	11.30	22.94
2K	6.87	12.91	28.19
4K	7.39	13.90	31.84
8K	7.68	14.46	34.04
16K	7.84	14.75	35.26
32K	7.92	14.90	35.90
64K	7.96	14.98	36.23
平均	7.38	13.89	32.06

实验结果分析：

1.在random测试中，基础指令集实现相比于C实现性能提升了**20%**，在medium测试中性能提升了**69%**，在large测试中性能提升了**88%**；

2.在random测试中，RVV实现相比于C实现性能提升了**85%**，在medium测试中性能提升了**222%**，在large测试中性能提升了**334%**。

汇报提纲

1. openEuler Embedded 介绍

2. openEuler Embedded 适配 VisionFive2

3. RISC-V分布式软总线

4. Musl libc的RVV优化

5. RISC-V 板卡对接 openEuler Embedded 流程与建议

RISC-V 板卡对接 openEuler Embedded 流程与建议

yocto-meta-openeuler bsp 目录下存放 openEuler Embedded 当前支持的 bsp 层，新增的 bsp 层应放在此目录下，以下对目录下各 bsp 层作简要介绍：

- **meta-openeuler-bsp**

  该层作为上游层与 openeuler 层之间的桥梁，提供统一的对 bsp 层的修改，层中对recipes的修改基于yocto工程中 bbappend 形式来实现。

- **meta-raspberrypi**

  此层提供树莓派开发板在硬件上的元数据文件，例如一些基本固件，与开发板硬件强相关的 recipes 都放在该层中。

- **meta-xxx**

  其它支持的 bsp 层，如 meta-rockchip、meta-visionfive2 与 meta-hisilicon。

RISC-V 板卡对接 openEuler Embedded 流程与建议

新增 BSP 层的目录层级一般分布如下：

- conf：存放一或多个 bsp_root_name.conf 文件；
- classes：存放硬件相关的类文件目录；
- recipes-bsp：存放启动相关固件的 recipes 目录；
- recipes-kernel：存放与内核相关固件 recipes 目录；
- recipes-*：其他的一些recipes目录，通常采用 bbappend 对源 bb 进行适配，实现适配硬件的功能；

RISC-V 板卡对接 openEuler Embedded 流程与建议

添加自定义的 BSP 层流程：

1. 初始化 yocto 构建环境；

2. 增加 BSP 对应的层；

```
# 生成一个默认层，可以自定义层路径，默认在当前目录生成
$ bitbake-layers create-layer /path/to/yocto-meta-openeuler/bsp/meta-bsp_name
# 添加层到bblayers.conf
$ bitbake-layers add-layer /path/to/yocto-meta-openeuler/bsp/meta-bsp_name
```

3. 根据需求编写 conf/machine/bsp_root_name.conf、bb、bbclass （bb、bbclass文件是可选项），通常需修改CPU、内核类型、根文件系统格式、设备树、内核模块等相关变量；

4. 修改构建目录中 conf/local.conf 中 MACHINE 变量的值为 conf/machine/bsp_root_name.conf 实际的值（不加.conf）；

5. 构建镜像

RISC-V 板卡对接 openEuler Embedded 流程与建议

添加 BSP 平台配置文件

oebuild 在构建时会解析 .oebuild/platform 下相应的平台配置文件，产生 compile.yaml 配置文件来完成构建操作。openEuler Embedded 适配 BSP 板卡，除了需要在 bsp 目录下新增相应的bsp层，还需要添加对应的平台配置文件 machine.yaml。

```yaml
type: platform

machine: starfive-dubhe

toolchain_type: EXTERNAL_TOOLCHAIN:riscv64

layers:
  - yocto-meta-openeuler/bsp/meta-visionfive2
```

visionfive2.yaml

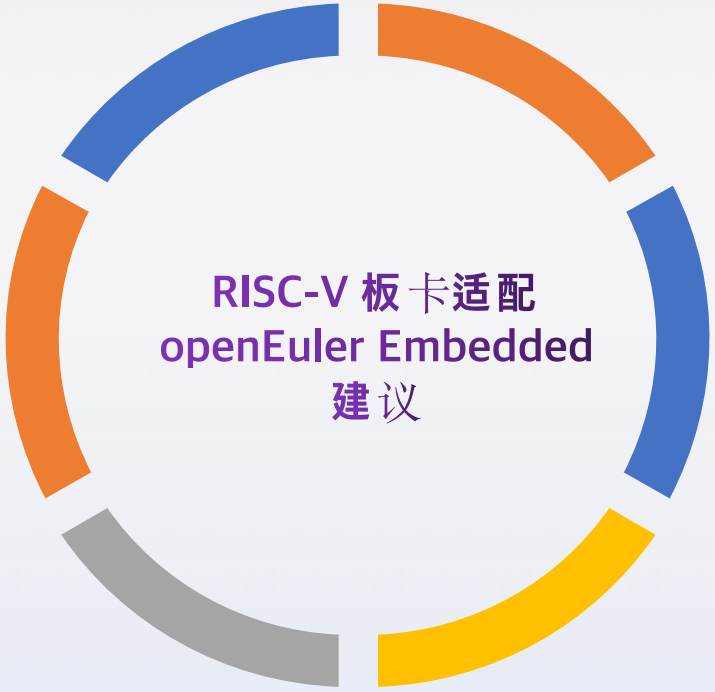
```yaml
build_in: docker
platform: visionfive2
machine: starfive-dubhe
toolchain_type: EXTERNAL_TOOLCHAIN_riscv64
repos:
  yocto-poky:
    url: https://gitee.com/openeuler/yocto-poky.git
    path: yocto-poky
    refspec: v3.3.6

  yocto-meta-openembedded:
    url: https://gitee.com/openeuler/yocto-meta-openembedded.git
    path: yocto-meta-openembedded
    refspec: dev_hardknott
local_conf: |
layers:
- yocto-meta-openeuler/bsp/meta-visionfive2
```

compile.yaml

RISC-V 板卡对接 openEuler Embedded 流程与建议

RISC-V 板卡适配建议



优先做好内核的适配工作

新增的 BSP 需要使用 openEuler 维护的内核，来自openEuler代码仓

进行相关固件的移植

opensbi、uboot及其他所需安装包均优先使用社区提供的源，需要提前进行验证和适配工作

测试工具链可用性

目前支持和使用的外部工具链有ARM、ARM64、RISCV64、X86四种，对应不同的平台处理器架构，若对工具链有特殊需求，需提前验证提供的工具链可用性


OpenEuler