鸿蒙 LiteOS-M 到 RISC-V 平台的移植

陈禹诚

yucheng.c@rioslab.org

RISC-V International Opensource Laboratory

Overview

- 背景知识
- 移植工作
- 测试
- 社会效益
- 结论

背景介绍：指令集架构

- 起源：IBM360(1964)
  - IBM公司公布了6款硬件实现不同的机器
  - 程序不需要修改都能运行
- 指令集架构是什么？
  - 是**计算机的软硬件接口**
  - 是一台计算机的抽象模型
  - 是计算机中最重要的接口之一
- 现状
  - 市场上PC，服务器市场被X86架构占领
  - 市场上嵌入式设备，手机，平板被ARM占领

- 总结
  - 商用闭源指令集架构目前主导处理器市场，从嵌入式设备到手机电脑，到服务器，基本上都在使用商用**闭源**指令集架构

RISC-V®

RISC-V: The Free and Open RISC Instruction Set Architecture

arm

背景介绍：RISC-V

- **RISC-V是什么？**
  - RISC-V（发音为“risk-five”）是一个基于精简指令集（RISC）原则的开源指令集架构（ISA）。
  - 该项目2010年始于加州大学伯克利分校，但许多贡献者是该大学以外的志愿者和行业工作者。

- **商用指令集架构有什么缺陷？**
  - 公司依靠售卖知识产权核的设计或者芯片来盈利，他们**不希望其他人能够自由的去用**指令集架构去做出硬件实现来争夺他们的市场。**阻碍创新。**
  - **极其复杂**。就算是属于精简指令集架构的ARM，单单是整数指令都超过了 600条(ARMv7)，即便法律上得到了ARM的授权，基于这个指令集架构设计一个具体的硬件实现门槛也是极高的
- **RISC-V作为开源指令集架构有什么好处？**
  - 能通过自由市场竞争**促进更多的创新与好的架构**的诞生
  - 能**共享好的核的设计，降低复用成本**，减少设计错误与漏洞后门
  - 能降低处理器成本，好处：例如降低成为物联网设备的门槛

PicoRIO

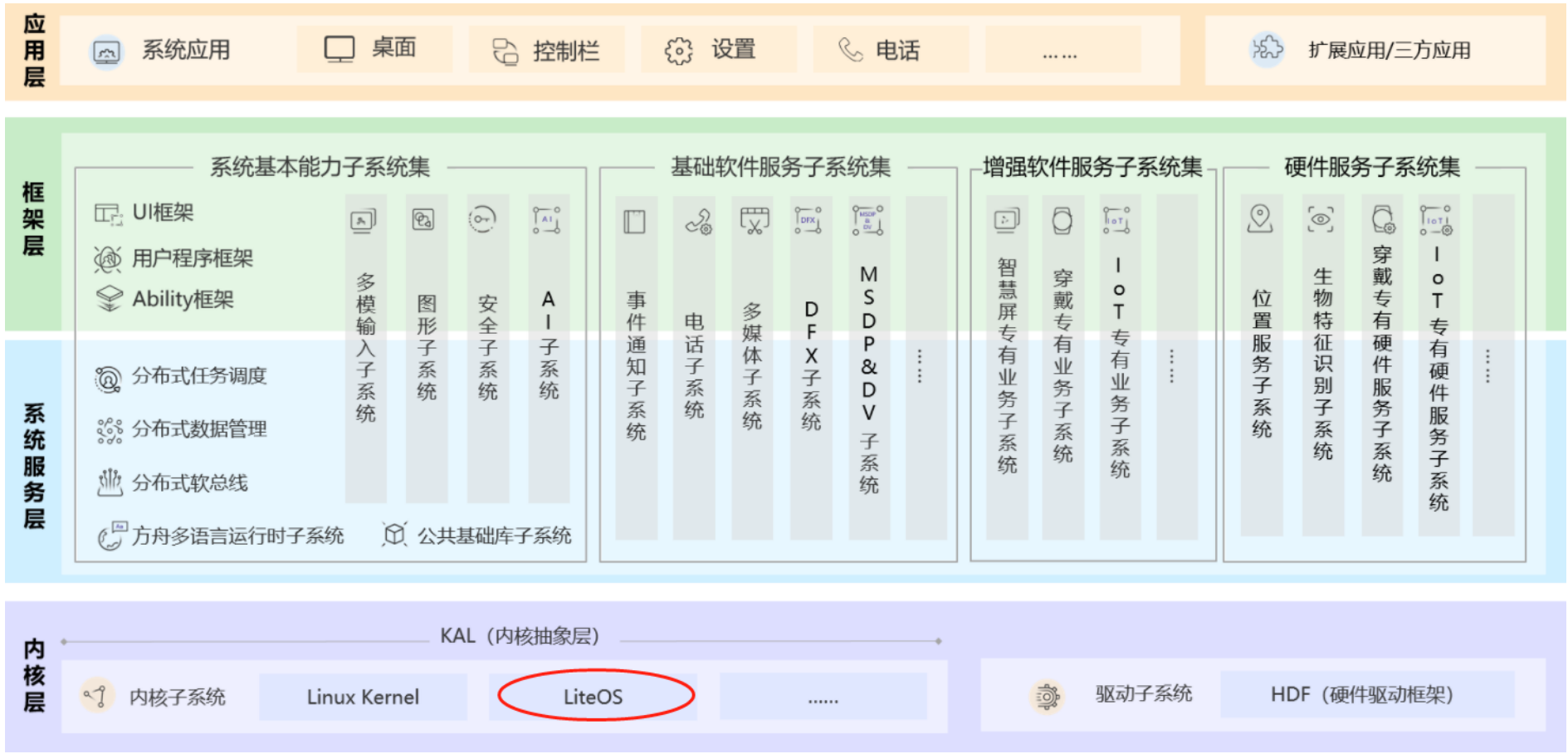
- PicoRIO是清华大学RISC-V国际开源实验室(RIOS)发布的微型计算机系统
- PicoRio从CPU设计，到PCB电路板设计，再到操作系统核心软件将全部开源

PicoRIO EVB

- PicoRIO EVB是PicoRIO试验阶段的一个评估板
- PicoRio EVB是 PicoRio 的一款测试评估板，支持RISCV IMC指令，有L1缓存 8KB，L2 缓存 256KB，支持 I2C，SPI，GPIO，UART 等 IO 端口。

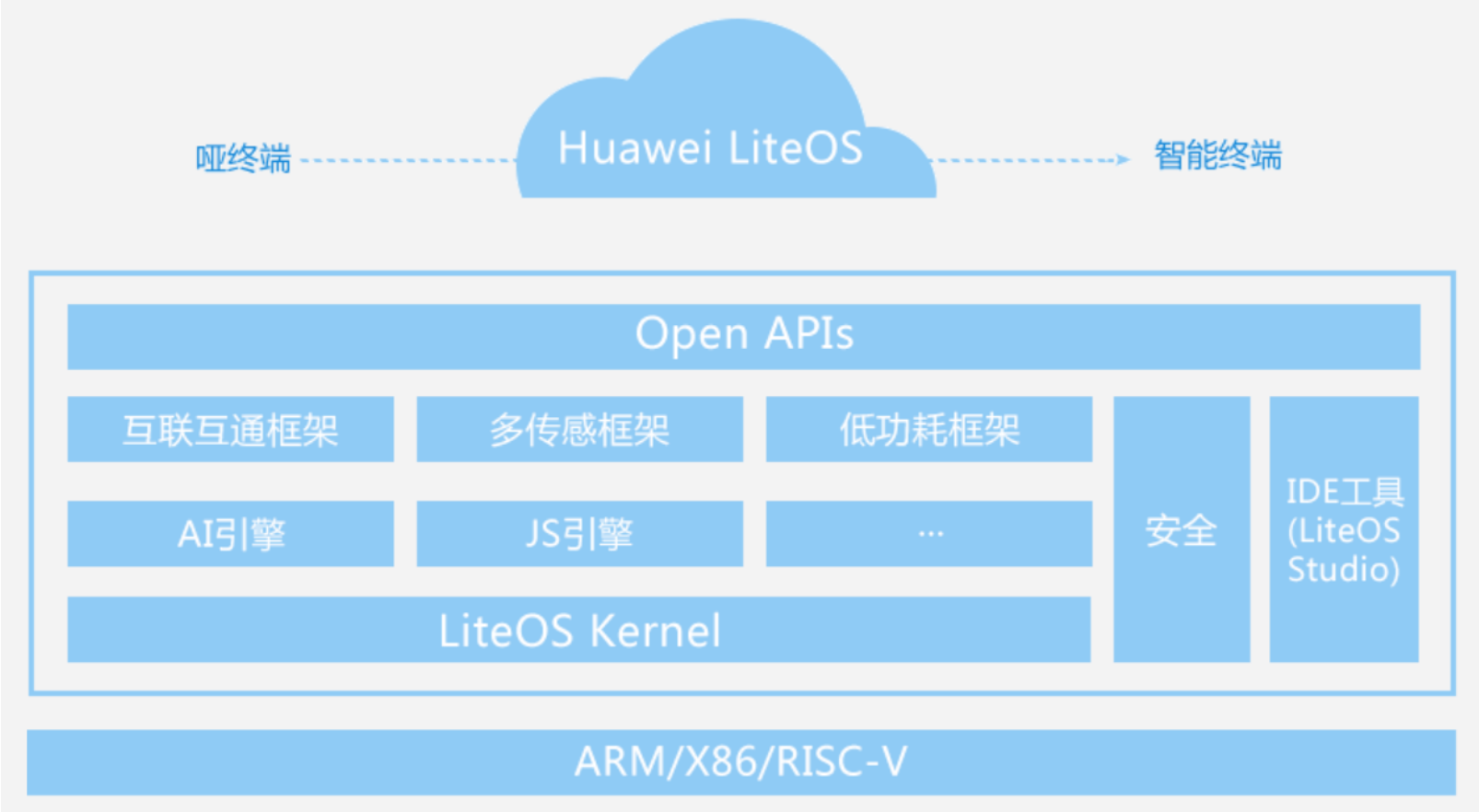
背景介绍：OpenHarmony开源项目

- OpenHarmony（鸿蒙）
  - OpenHarmony是开放原子开源基金会（OpenAtom Foundation）旗下开源项目，由华为捐赠。定位是**一款面向全场景的开源分布式操作系统**。



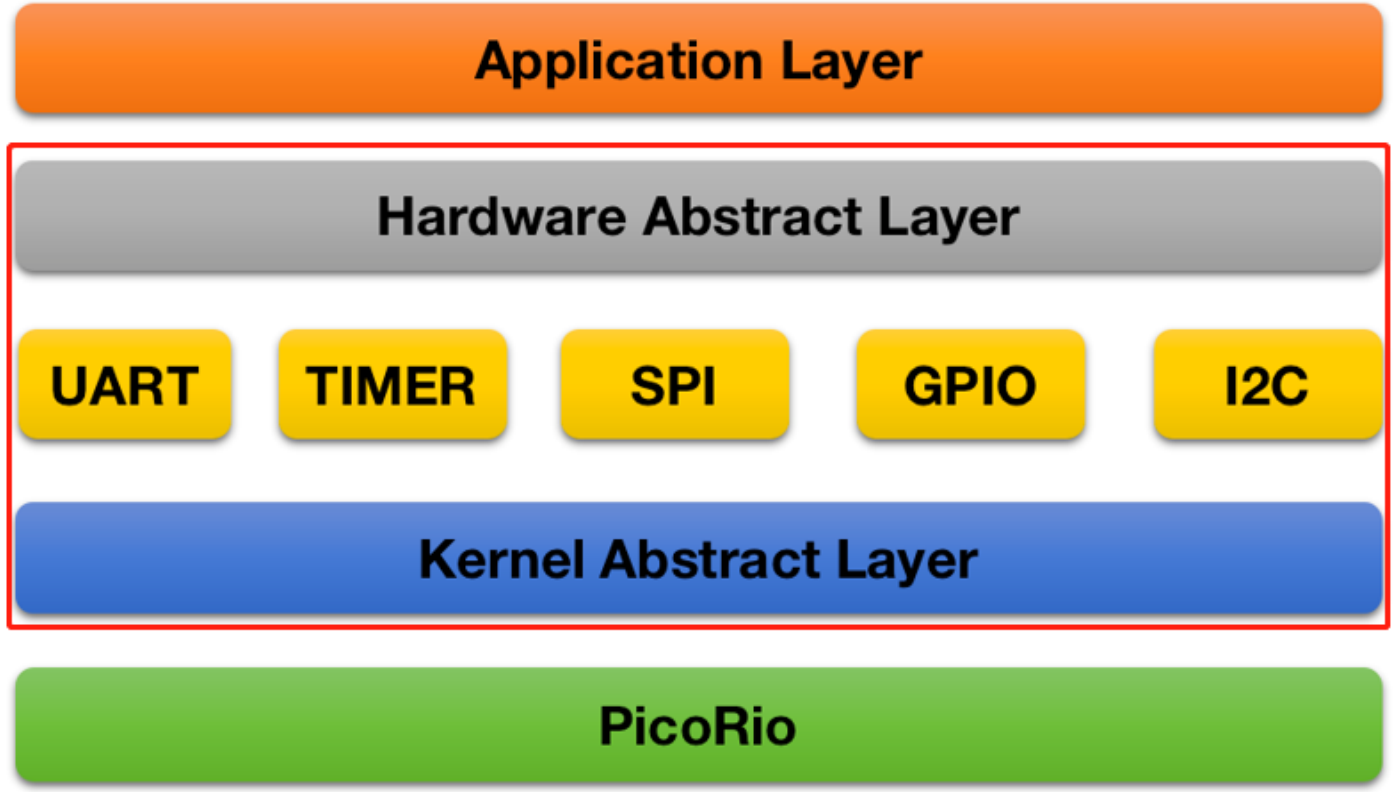
背景介绍：LiteOS-M

LiteOS-M是Openharmony的子项目，为上层提供内核服务接口。



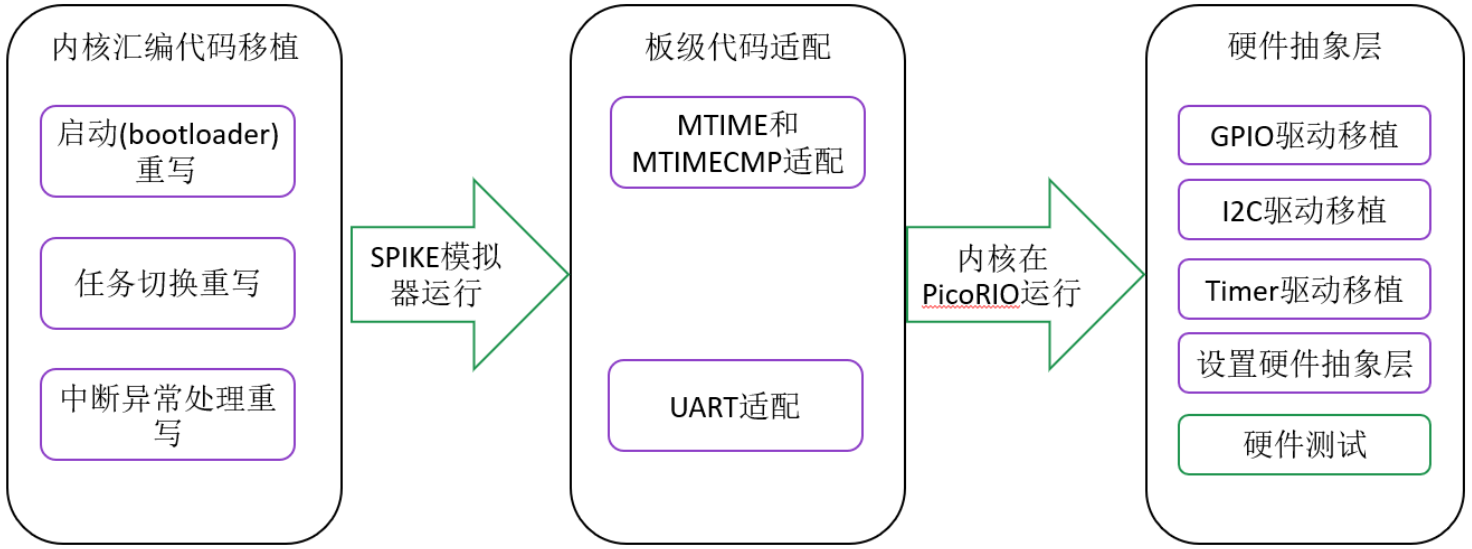
一句话总结工作内容

将开源的 OpenHarmony的LiteOS-M操作系统移植到RISC-V PicoRIO EVB平台上。



移植工作

- 内核汇编代码移植
  - 阶段目的：**操作系统能够正常在软件模拟器上执行**
- 内核板级代码适配
  - 阶段目的：**操作系统能跑到真实的芯片上，并且能够通过UART串口输出消息**
- 硬件驱动移植
  - 阶段目的：**各种串口，各种硬件IP，各种外设能够使用（cheers！）**



移植工作：内核汇编移植

- 操作系统源代码分为C代码和汇编代码两部分
  - C代码可以通过RISC-V编译工具链形成二进制文件
  - 汇编代码没有RISC-V版本
- 所有内核汇编代码需要用RISC-V汇编重构！

- 阶段目的：**操作系统能够正常在软件模拟器上执行**

移植工作：内核汇编移植

- Bootloader
  1. 设置 mstatus 的MPP 字段为 3
     - LiteOSM是轻量级物联网操作系统，全程在机器特权级下运行。故MPP 应当为 3 确保 mret 后特权级仍然为机器特权级。
  2. 清空 mie 寄存器
     - 初始状态下是没有外部中断的，要确保 mie 被初始化。
  3. 设置 mtvec 的值指向**中断向量表**的地址
     - 只有设置了 mtvec 才能正常触发中断以及异常处理
  4. 设置**栈指针寄存器** SP 的值为栈基址，建立 C 语言执行环境，栈基址是在链 接脚本文件定义的栈段确定的
  5. 跳转到 LiteOS 的 main 主函数，启动汇编代码部分结束，main 函数后由C语 言代码接管

```
.extern memset
.global _start
.section .start.text
.align 4
_start:
    li    t0, RISCV_MSTATUS_MPP
    csrw  mstatus, t0
    csrw  mie, zero
    la    t0, TrapVector
    csrw  mtvec, t0       # direct mode

    .option push
    .option norelax
    la    gp, __global_pointer$
    .option pop

    la    t0, __bss_start
    la    t1, __bss_end
2:
    sw    zero, 0x0(t0)
    addi  t0, t0, 0x4
    bgtu  t1, t0, 2b

    la    sp, __start_and_irq_stack_top

    j  main
```

移植工作：内核汇编移植

- 任务切换机制
  - 当内核初始化完毕，LiteOS 会通过 LOS_StartToRun进入任务调度模式开始执行任务。
  - 在运行 runTask 的 TCB 对应的任务，由于时间片耗尽，或是调用了任务休眠函数等原因，LiteOS 会调用 LOS_Schedule 函数将当前 runTask 换掉。
- RISCV 汇编实现 LOS_StartToRun
  1. 初始化 g_taskScheduled 为 1 来开启调度
     - 因为如果 g_taskScheduled 为 0，LOS_Schedule 函数只会更新 newTask
  2. **初始化runTaskTCB**,在汇编中将 runTask 赋为 newTask，包括上下文寄存器
     - newTask 会在任务创建时就维 护好
     - 在创建任务初始化任务栈空间时，会把寄存器上下文存在栈中
  3. 切换 TCB 以及设置 TCB 过程中，要确保中断是关闭的
     - 可以通过清除 mstatus 的 MIE 字段来关闭中断

移植工作：内核汇编移植

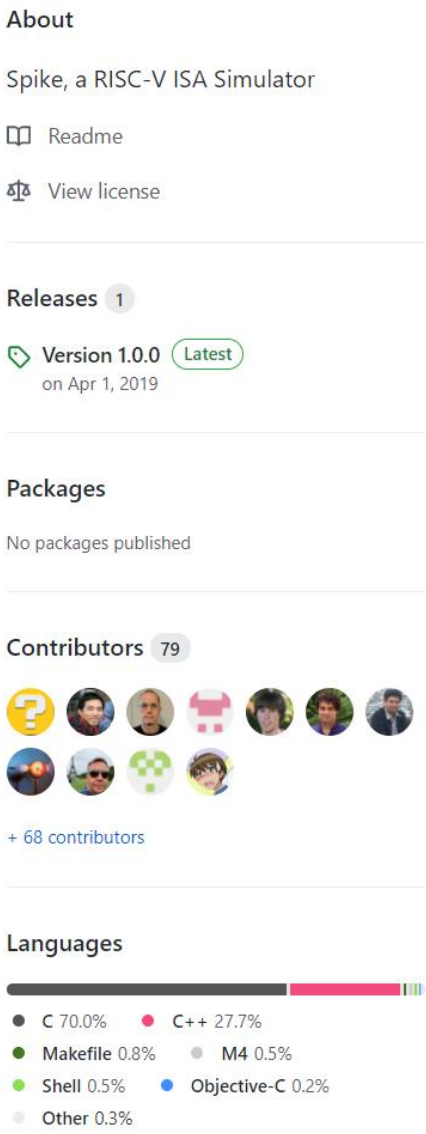
- **RISC-V汇编实现上下文切换**
  1. 将包括 mstatus 和 mepc在内的上下文寄存器保存
     - mepc 设置成 ra寄存器的值，也就是进入上下文切换函数前的原任务的地址
  2. 设置TCB的运行状态字段
  3. 将newTask 拷贝覆盖 runTask
  4. 接着获取原栈指针后**恢复原** newTask 的上下文
  5. 恢复 mepc 的之后调用 mret 指令，**pc 指针**将继续执行原 newTask 的 mepc 地址之后的指令

移植工作：内核汇编移植

- RISC-V 汇编实现中断处理
  - 出现中断时，pc 会跳转到 mtvec 地址，执行中断处理的汇编函数（中断向量表）
  - 中断向量表会分类讨论mcause 的 Interrupt 字段两种中断类型
    ① Interrupt 值为 0，意味着这是异常
       1. 读取当前寄存器的值存在栈中保留异常现场
       2. 调用LiteOS的OsExcEntry输出在栈中保留的寄存器值与TCB值
       3. 输出完异常进入执行对应错误的中断处理向量
    ② Interrupt 值为 1，意味着这是中断
       1. 直接进入中断向量进行处理

移植工作：软件ISA模拟器SPIKE拟运行

- Spike是什么？
  - 是RISC-V开源CPU软件模拟器
- 为什么要用Spike？
  - DEBUG！！！
  - Spike能够在调试模式逐条走汇编，并随时输出寄存器的值
  - 直接烧录进板子如果有BUG，看不到任何反馈消息



移植工作：内核板级代码适配

- 内核板级代码适配主要工作是将内核时钟频率和 PicoRio EVB 的时钟频率对齐。并且为了确认板级代码适配成功，需要适配 UART 来输出 PicoRio EVB 的打印信息。

- 阶段目的：**把LiteOS-M跑到真实的芯片上，并且能够通过UART串口输出消息到PC上**

移植工作：内核板级代码适配

- MTIME寄存器与MTIMECMP寄存器
  - 这两个寄存器是用来触发时钟中断的（mcause exception=7）
  - 一般地，MTIME会随着cycle数线性增长
  - 当MTIME大于等于MTIMECMP的时候，触发时钟中断
  - 例如：
    - 希望20000个cycle发生一次时钟中断
    - 设MTIME和cycle是1：1增长
    - 初值可以设MTIME=0，MTIMECMP=20000
    - 每次中断MTIMECMP增加20000

移植工作：内核板级代码适配

- UART适配
  - 如果不配上UART串口输出，操作系统就是个“哑巴”，无法输出信息。无法得知是否在运行，是否进trap了，是否能成功切换等等。无法调试。
  - 调试需要UART，UART需要调试
  - 本文UART 采用轮询的方式对外进行输出
  - 每输出一个字符 ch，驱动函数会将字符写到UART_THR的寄存器地址上，然后轮询读取UART_LSR寄存器的状态值， 如果该状态的 TEMT 和 THRE 字段同时为 1，说明输出成功，结束轮询，可以继续输出下一个字符。

移植工作：内核板级代码适配

- 在PicoRio EVB上运行LiteOS-M内核
  - **配好UART后**，用spi将LiteOS-M导入PicoRIO EVB
  - 将UART的USB一端连向电脑，配置电脑minicom串口波特率
  - **打开minicom命令行**监视输出信息流来判断是否成功，或者是通过输出信息调试。

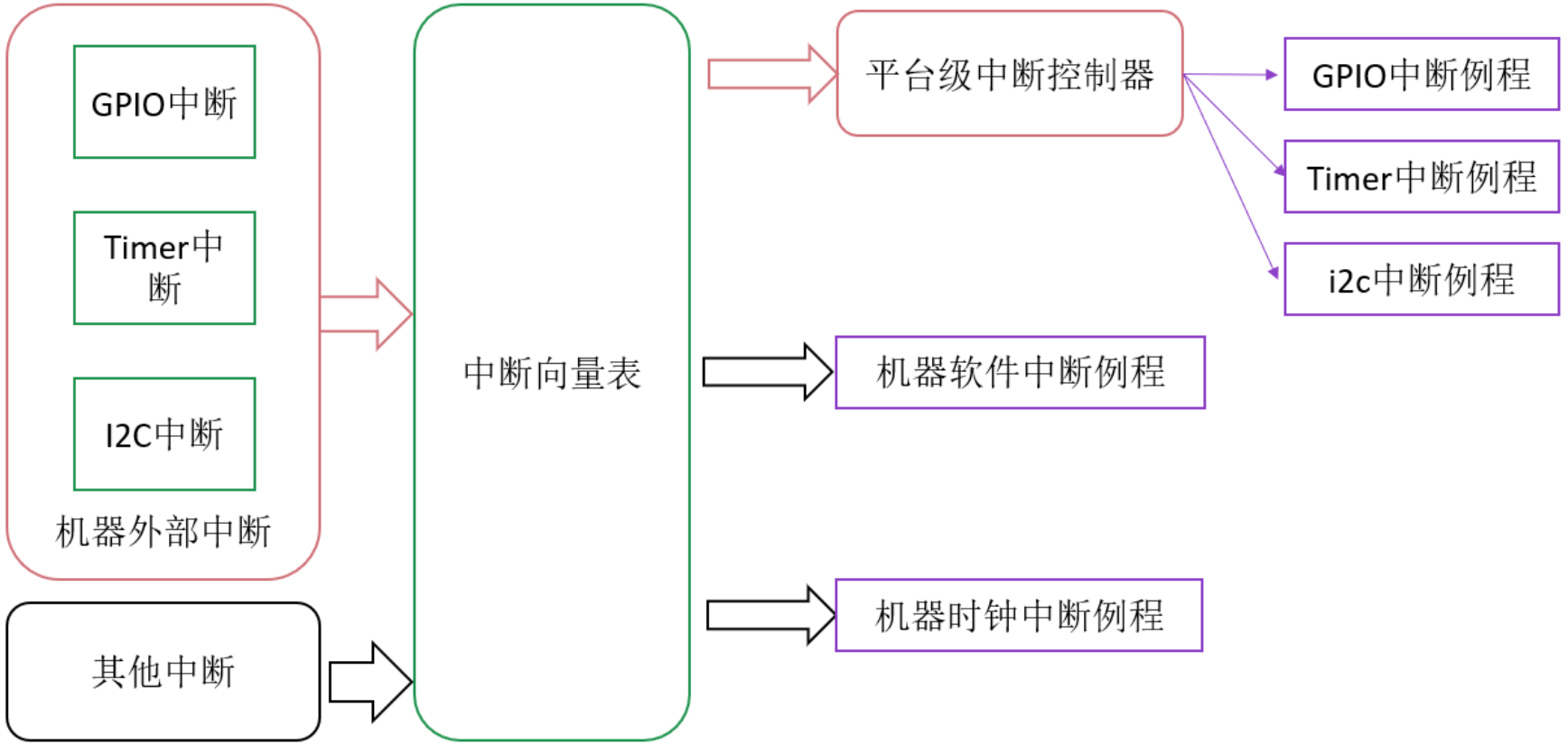
移植工作：驱动代码移植

- **内核代码能够在 PicoRio EVB 平台上正常运行**后，要想提供更多的硬件服务，需要移植硬件驱动到 LiteOS-M 上。本文在驱动代码移植的工作主要为移植 timer， GPIO，i2c 到 LiteOS-M 上。

- 阶段目标：
  1. **硬件驱动正常工作**
  2. **为LiteOS-M建立硬件抽象层，管理多个硬件，并为上层提供一致性接口**

移植工作：驱动代码移植

- 平台级中断控制器(PLIC)
  - PLIC是RISC-V硬件独有的结构

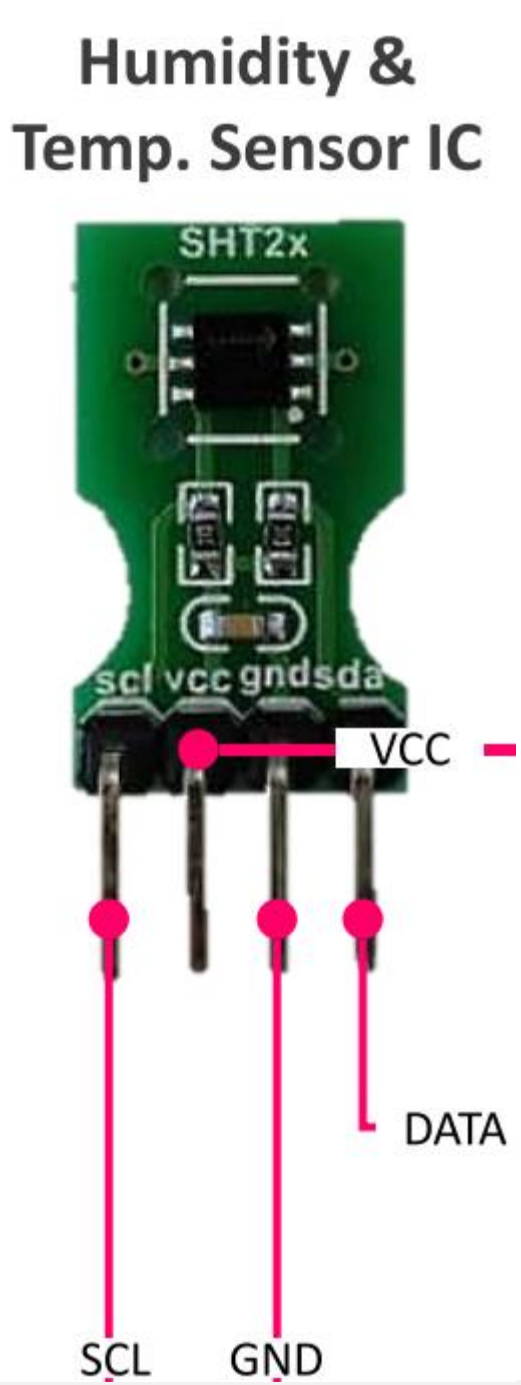


移植工作：驱动代码移植

- Timer
  - PicoRio EVB采用的Timer含有8个相同的独立的计时器
  - 每个计时器有独立的配置，使能地址，通过读写地址来控制
  - 本文定义了一个数据结构 timer_cfg_struct 来完成寄存器的配置，内含定时器时间，PWM，计时器模式等成员变量，以及配置函数 rvhal_timer_config，将数据 结构和计时器 id 传入配置函数，会直接将成员变量的值写入对应寄存器
  - 本文对计时器的开启与关闭的实现是通过更改 TimerNControl 的第 1 个使能 字段进行实现。

移植工作：驱动代码移植

- I2C
  - I2C(I Squared C) 是一种同步多主多从**串口通讯协议**
  - I2C 驱动在初始化时会向 PLIC 注册 I2C 中断处理例程
  - 当上层应 用想要使用 I2C 进行通讯时，可以调用 I2CTransfer 函数
  - I2CTransfer 函数会解除对 I2C PLIC 中断的屏蔽，指定 I2C 收发 buffer 地址，并使能 I2C 模块
  - 触发 I2C 中断时会通过寄存器对应字段判断是发送，还是接收数据。判断 完后会通过读写指定的收发 buffer 来完成数据的传输



移植工作：驱动代码移植

- GPIO
  - GPIO(General Purpose Input/Output) 是微控制器和其他电子设备进行交流的标准接口。**GPIO 可以与例如传感器，灯泡，显示器等设备相连**，进行信息传输，控 制等活动。
  - 本文实现了一些GPIO读写接口。GPIO 的每个 pin 有**输入，输出两种状态**，本文实现了 GPIOSetDir 与 GPIOGetDir，可以用 GPIOSetDir/GPIOGetDir 来设置或读取 pin 的状态。用杜邦线将一 个输出 pin 和输入 pin 相连，当用 GPIOWrite 写输出 pin 时，可以在输入 pin 用 GPIORead 读到写的值。
  - 本文实现了 **GPIO 的 PLIC 中断**，一共有五种触发中断的方式，分别是水平和边缘两种触发模式和高电压和低电压两种触发条件的组合，外加一种无论是高电压边缘还是低电压边缘都会触发的模式。

移植工作：硬件抽象层设计

- LiteOS-M目前没有HAL（硬件抽象层）
- 什么是硬件抽象层？
  - 统筹管理各个硬件驱动，包括同类不同种的硬件驱动
  - 每个硬件驱动是该硬件抽象类的实例

测试内核

- 任务切换测试

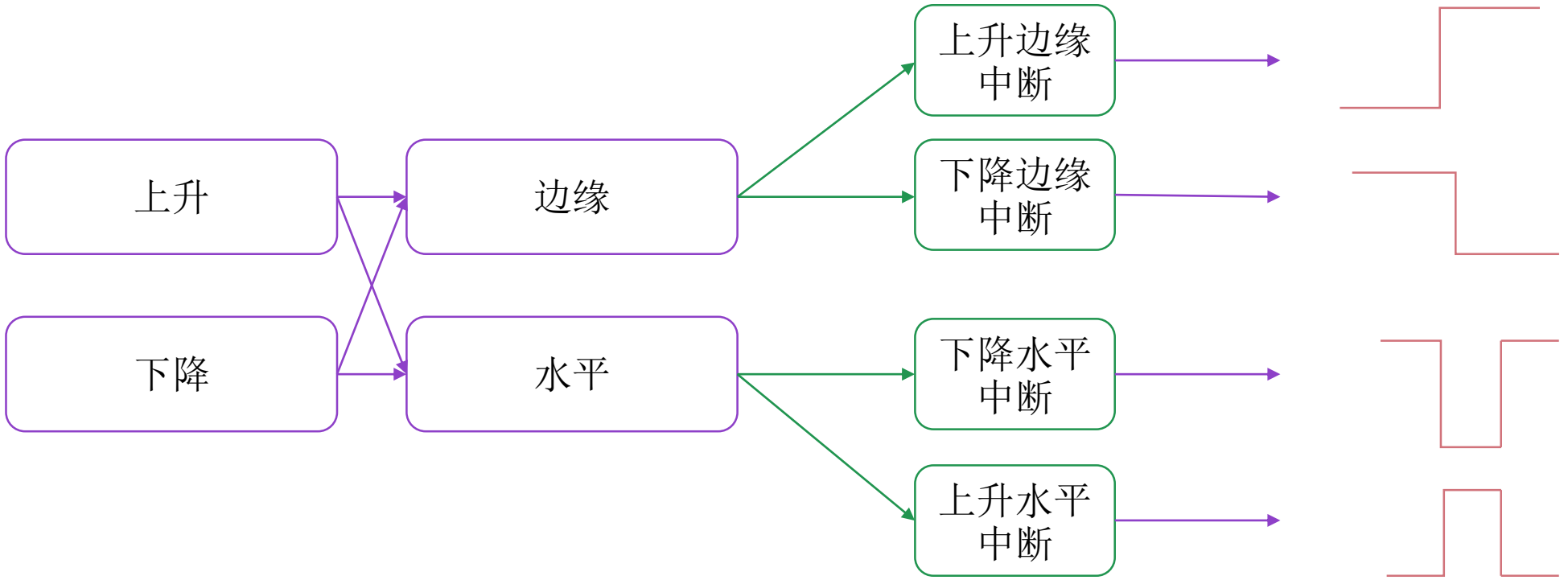
```
VOID TaskSampleEntry1(VOID)
{
    while(1) {
        LOS_TaskDelay(1000);
        printf("TaskSample1 helloworld...\n\r");
    }
}

VOID TaskSampleEntry2(VOID)
{
    while(1) {
        LOS_TaskDelay(500);
        printf("TaskSample2 helloworld...\n\r");
    }
}
```

GPIO读写测试方法

```
// 设置连接的两个GPIO Pin的输入口和输出口
GpioSetDir(GPIO_TEST_PIN_IN, GPIO_DIR_IN);
GpioSetDir(GPIO_TEST_PIN_OUT, GPIO_DIR_OUT);
uint16_t in_value, out_value;
// 测试设置是否成功
GpioGetDir(GPIO_TEST_PIN_OUT, &out_value);
GpioGetDir(GPIO_TEST_PIN_IN, &in_value);
printf("out dir: %u\n\r", out_value);
printf("in dir: %u\n\r", in_value);
// GPIO读写测试
```

```
for (i = 0; i < GPIO_TEST_TIMES; i++) {
        out_value = i & 1;
        GpioWrite(GPIO_TEST_PIN_OUT, out_valu
        for (j = 0; j < 10000; j++);
        GpioRead(GPIO_TEST_PIN_IN, &in_value)
        if (out_value != in_value) {
                err = 1;
                break;
        }
}
```

GPIO中断测试方法

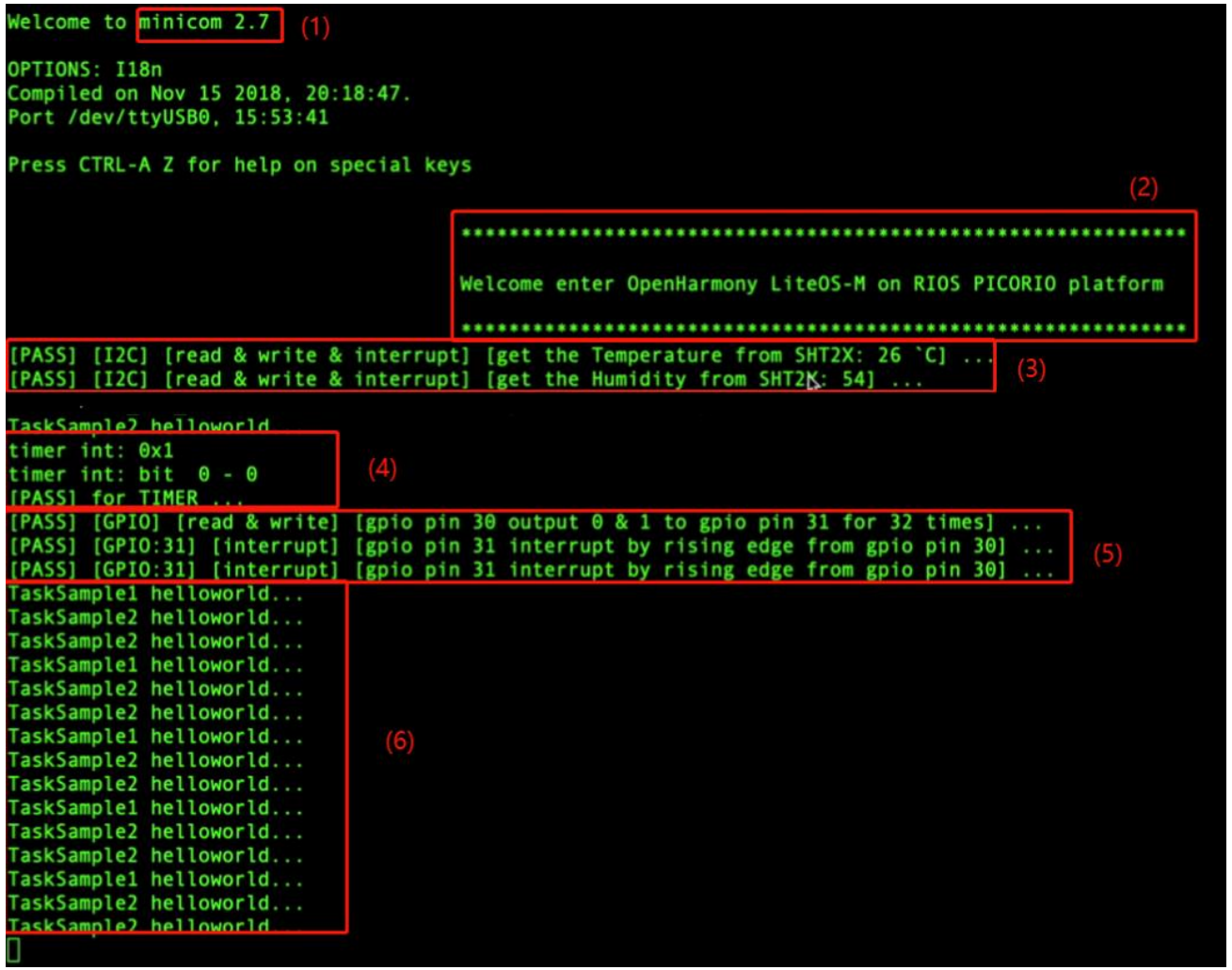


I2C测试：数字温湿度传感器 SHT2x

```c
unsigned char SHT2x_MeasureData(void)
{
    struct I2cDev *dev = I2cOpen(0);
    if(!dev){
        printf("[FAIL], I2C DevNum not exist!\n\r");
        return;
    }
    snt16 temp, humi;
    unsigned char error = 0;
    short temperature, humidity;
    error = SHT2x_MeasureHM(TYPE_T_MEASUREMENT_HM, &temp, dev);

    if (0 == error)
    {
        temperature = (short)floor(SHT2x_CalcTemperatureC(temp.u16));
```

1. 打开I2C硬件抽象层实例
2. 用I2C发送测温度指令给SHT2x
3. 用I2C接收测试数据，按照公式算出温度

测试结果

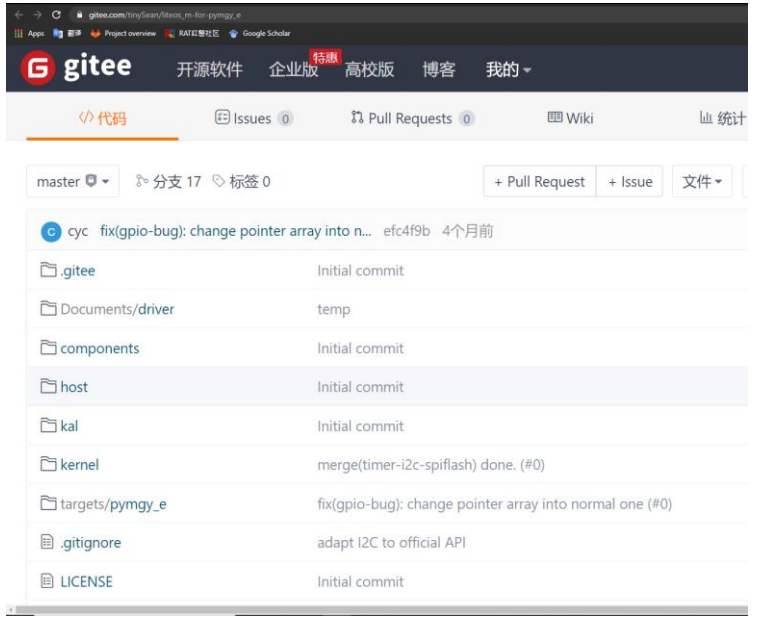


(1) minicom将内核输出信息打印在信息控制台
(2) 内核欢迎信息，代表以及进入内核C语言执行环境
(3) I2C温湿度传感器结果：温度26度，湿度54
- 当时在空调房，比较干燥

(4) timer定时器中断触发
(5) GPIO读写测试与GPIO中断测试
(6) 内核任务切换测试

社会效益

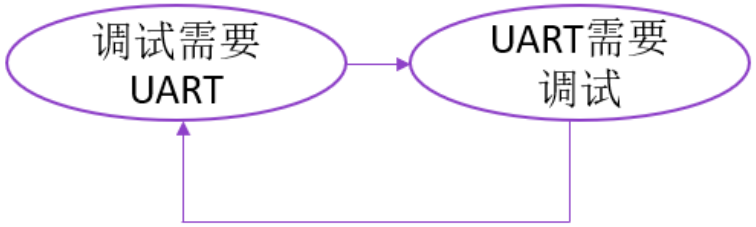
- 睿思芯科的Pygmy-E，Pygmy ES1Y系列芯片采用了本文的LiteOS-M作为其芯片配套的操作系统基础设施
- https://www.rivai.ai/product.php?id=21
- 电子发烧友，知乎等平台上相关教学机构采用本文的工作进行嵌入式编程，单片机开发板教学
- https://zhuanlan.zhihu.com/p/355441774
- https://bbs.elecfans.com/jishu_2040128_1_1.html



RIOS

总结

- 这篇工作，挑战性体现在哪？
  - Bootloader, 上下文切换, 中断异常处理等**内核汇编代码需要用RISC-V汇编重写**
  - 板级代码适配过程由于没有UART无法打印debug信息，**整个阶段需要盲调**
  - 硬件驱动需要实现**PLIC机制**，各个**硬件驱动**以及其测试，**硬件抽象层设计**



- 这篇工作，意义和贡献是什么？
  - 进一步完善 RISCV 开源社区的软件基础设施建设，加速 RISCV 指令集架 构的软件生态发展。
  - 增加 OpenHarmony 支持的硬件平台。目前 OpenHarmony 的 LiteOSM 支持 的是一些ARM指令集架构下的目标芯片。将 LiteOS-M 移植到RISC-V的硬 件平台下能够扩大 OpenHarmony 开源操作系统的运用范围。
  - **开源，免费的软硬件解决方案权益不被公司控制，也不会被国家所垄断，能允许人们自由更改用于商业，教学，科研等。软硬件开源方案能够产生值得提倡的社会效益。**

Comments?