基于RISC-V处理器嵌入式教学与应用实践

林金龙，何小庆

2025年07月18日

北京大学

内容提要

- RISC-V 应用现状
- 课程介绍
- RV32教学
- RV64教学
- 应用与实践
- 《深入理解RISC-V程序开发》（第二版）
- 小结

RISC-V 应用现状

- 市场研究机构Counterpoint Research预测：2030年RISC-V芯片出货量将突破160亿颗；RISC-V在IoT市场的出货量将增至108亿颗，全球汽车RISC-V芯片预计出货量将突破 25亿颗。
- 凭借开源开放、精简灵活的商业模式和技术特点上的优势，RISC-V的崛起已是大势所趋，中美科技竞争加速中国芯片企业采用RISC-V替代ARM架构。
- 嵌入式世界是RISC-V的主战场，百万级应用落地，生态日臻成熟：从物联网、工业控制到智能汽车，RISC-V MCU 无处不在！更高性能的处理器持续涌现，高算力嵌入式AI应用蓄势待发！

RISC-V 普及的瓶颈

- 总体上RISC-V 处理器在嵌入式设备、开发者和开发生态以及高校课程和电子/计算机/物联网大赛等行业应用中普及率不高，开发者熟悉度不强。
- 通用性的嵌入式RISC-V 处理器品类非常少，软件和应用生态建设以及开发的便捷和成熟性与Arm 生态有相当的距离。
- 车规级RISC-V的设计在积极推进，规模量产和装备还待时间，应用集中在车身控制和传感器方面。
- 高性能嵌入式RISC-V 处理器不断有新品发布，但作为主核未见有市场影响力的终端产品。

内容提要

- RISC-V 应用现状
- 课程介绍
- RV32教学
- RV64教学
- 应用实践
- 《深入理解RISC-V程序开发》（第二版）
- 小结

课程概述

名称： 嵌入式微处理器系统

性质：专业必修

课时： 48

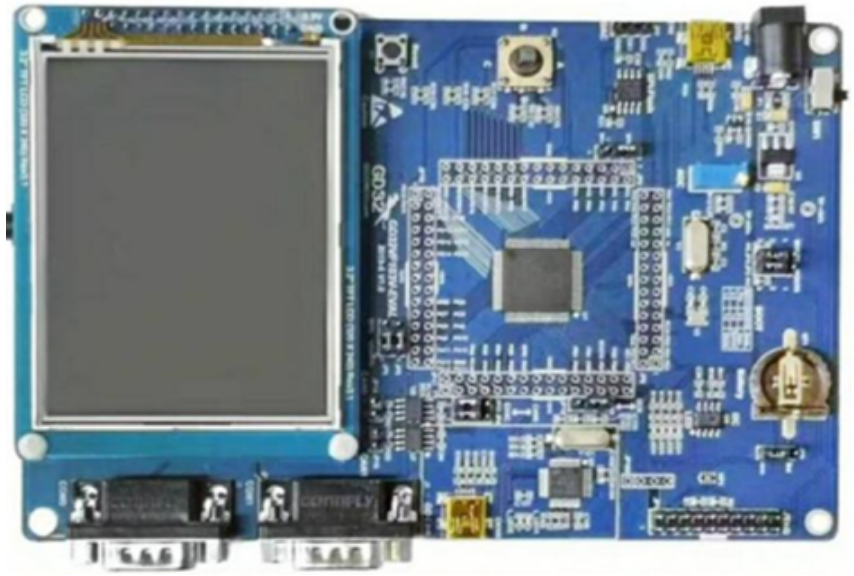
开课时间：研一第一学期

课程内容

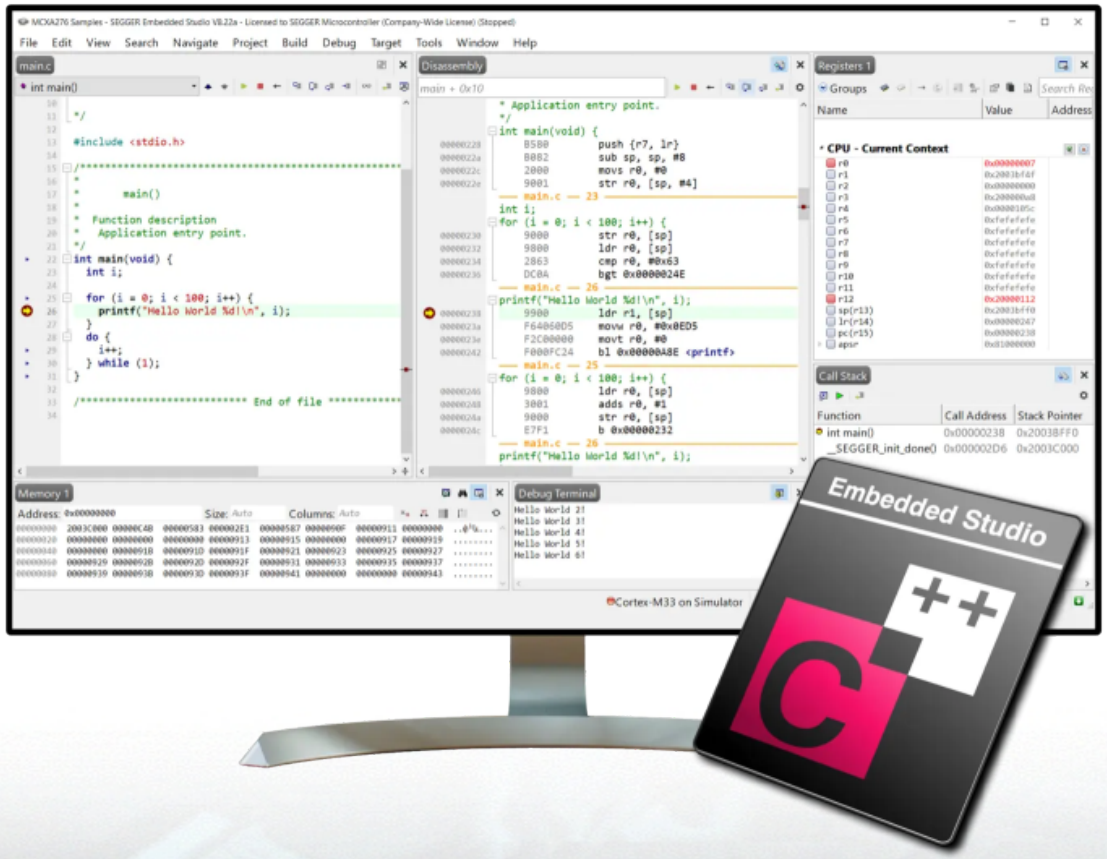
内容	知识点
处理器程序开发基础	处理器结构，指令执行，指令与程序，程序开发方法和开发工具
ARM指令与汇编程序	ARM家族，ARM9内核结构、异常模式和存储系统，ARM指令，ARM 汇编(arm asm / gcc asm)语法和程序设计，启动程序
ARM C/C++程序开发	函数调用，异常服务程序，存储管理，外设访问，FreeRTOS
ARM程序优化	程序优化定义与方法，运行时间优化，存储资源优化，能耗优化
RISC-V 程序开发	RISC-V概述，RV32 与 Cortex-M对照，GD32VF103，RV64系统软件环境，TH1520
多核程序开发	并行计算，同构多核处理器，异构多核处理器，OpenMP
GPU程序开发	GPU ，CUDA，CUDA 编程
应用实践	FreeRTOS移植，嵌入式人工智能示例

RISC-V教学平台

GD32VF103-EVAL



SEGGER Embedded Studio



北京大学

RV64 教学平台

LicheePi 4A(TH1520)

软件开发工具

软件包	组件	说明
develop tools	开发工具	gcc，g++, perl，python3 等
debug tools	调试工具	gdb、gdbserver、strace、glibc-mtrace、trace-cmd、memtool、kmod、binutils、systemd-analyze、kdump、procps 等
soc drivers	芯片设备驱动	GPU, VPU, ISP, NPU, DPU 以及各类外围设备驱动
system tools	系统工具	busybox，msfs，adb, util-linux, insmod, modinfo 等
connection tools	连接工具	wpa-supplicant、bluez5
u-boot	系统 bootloader	用于启动 Linux 系统
opensbi	RISC-V 的 sbi 镜像	由 RISC-V 的 SBI 标准实现的在机器态提供服务的镜像
kernel	操作系统内核	基于 Linux 5.10 以及玄铁 C910 的 CPU 扩展
security	安全体系	基于 OP-TEE 的安全运行环境以及安全应用等安全体系

北京大学

内容提要

- 课程介绍
- RV32教学
- RV64教学
- 应用与实践
- 《深入理解RISC-V程序开发》
- 小结

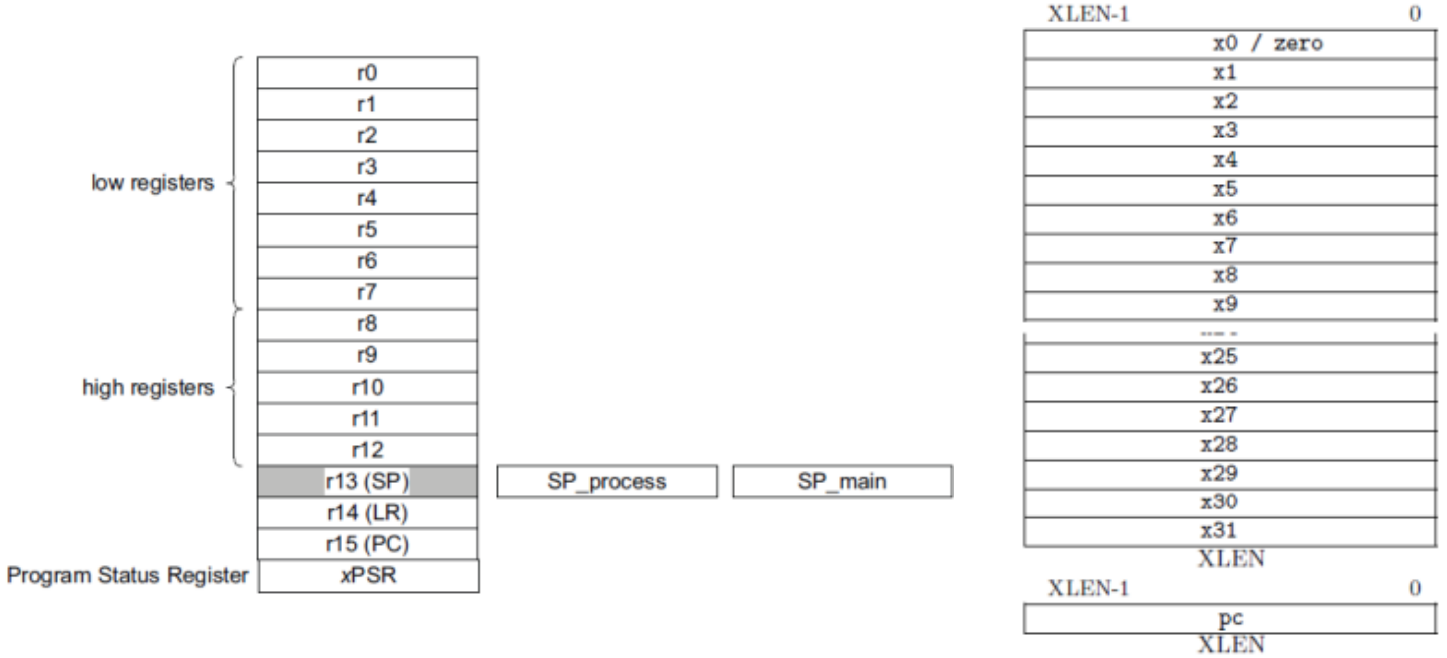
- RV32 与 Cortex-M对照
  - 寻址空间类型
  - 状态和模式
  - 异常进入和退出
  - 异常响应
  - 指令
  - 调用规范
- RV32 指令集模块选择分析
  - i
  - im
  - imc

RV32 教学-对比

寻址空间类型

Cortex-M3　　RV32

通用寄存器



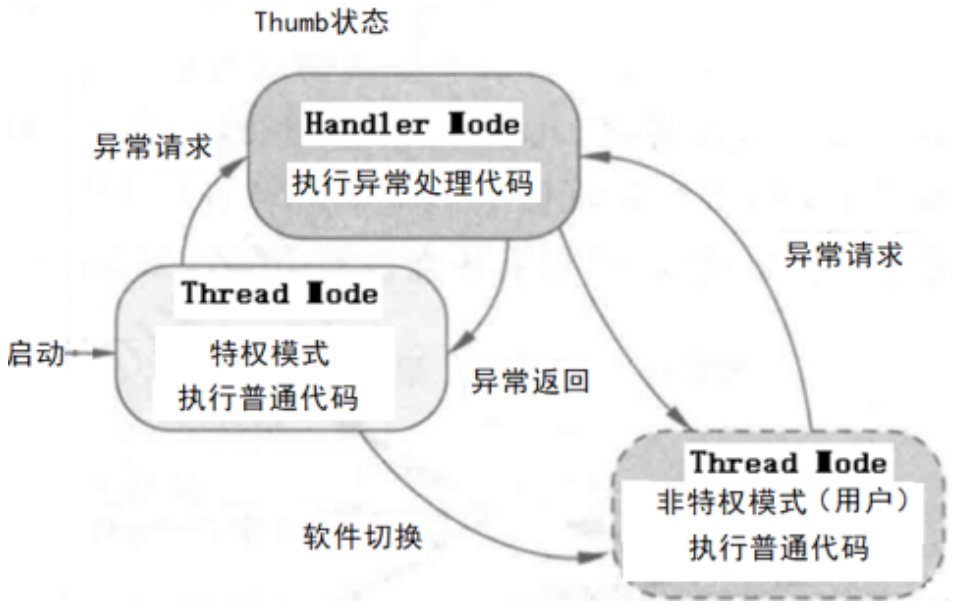
Cortex-M3　　RV32

《Cortex™-M3 Technical Reference Manual》,Revision r2p0，ARM

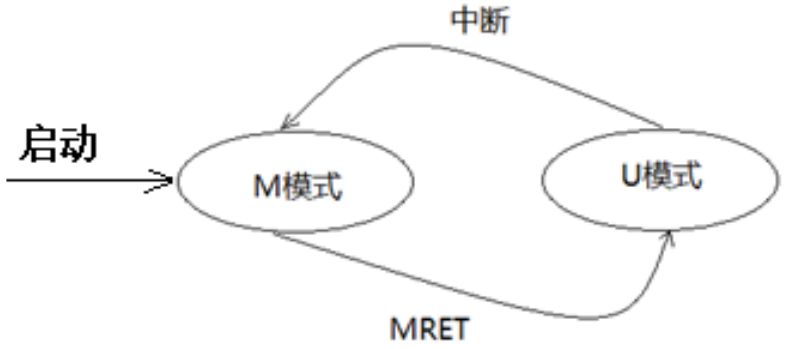
《The RISC-V Instruction Set Manual, Volume I：Unprivileged ISA》， Ver. 20190608

北京大学

模式和状态



Cortex-M3

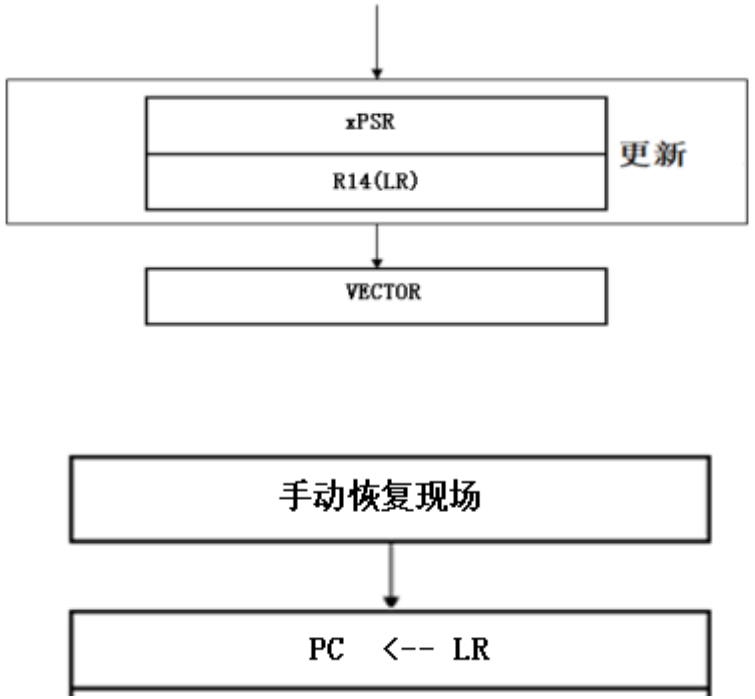


RV32

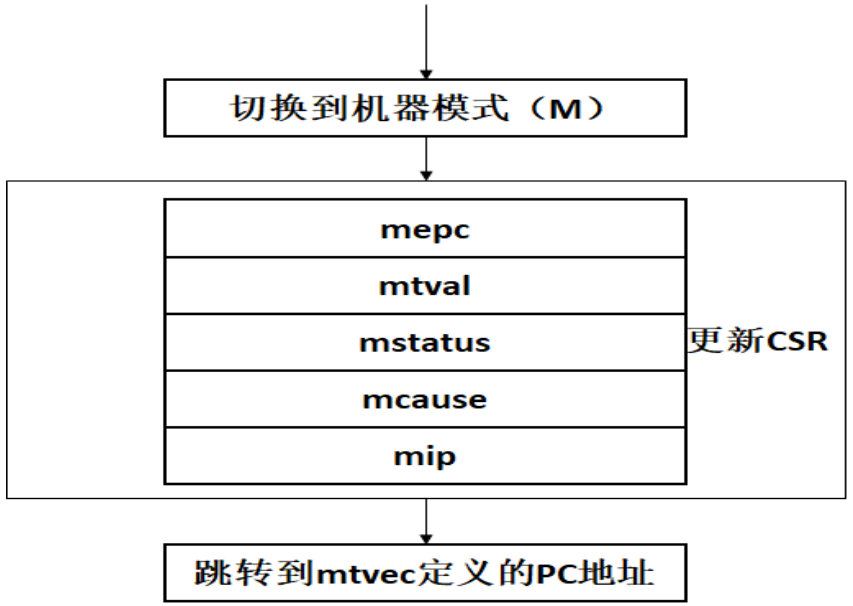
《Cortex™-M3 Technical Reference Manual》,Revision r2p0，ARM

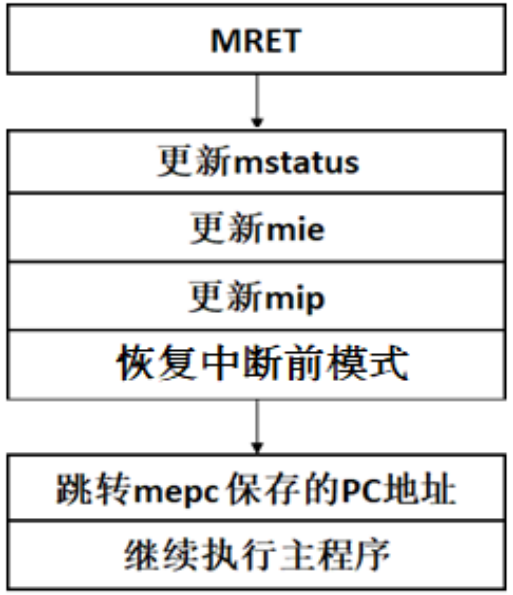
《The RISC-V Instruction Set Manual, Volume I：Unprivileged ISA》， Ver. 20190608

进入和退出异常



Cortex-M3



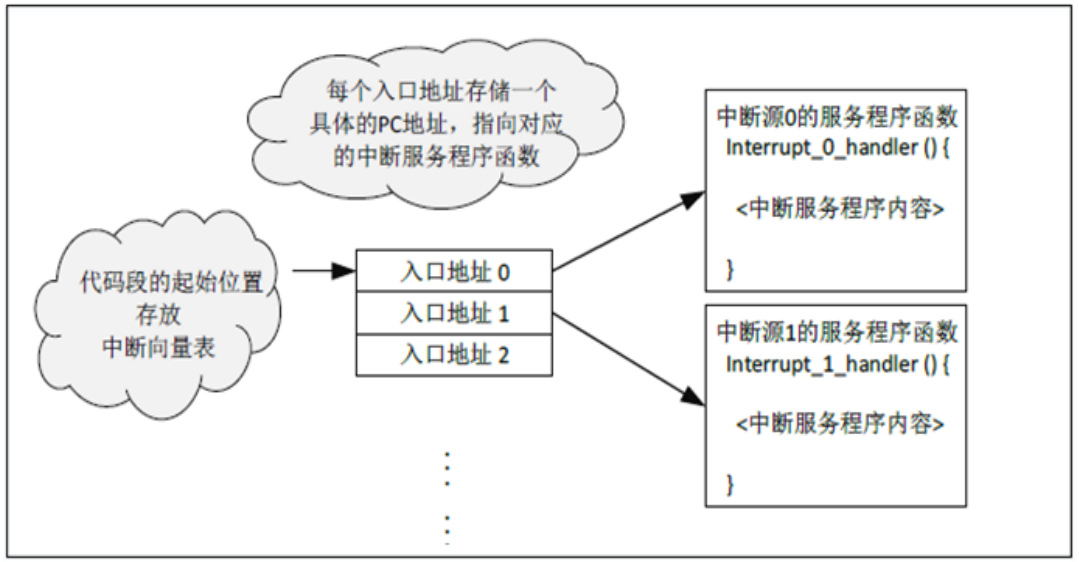


RV32

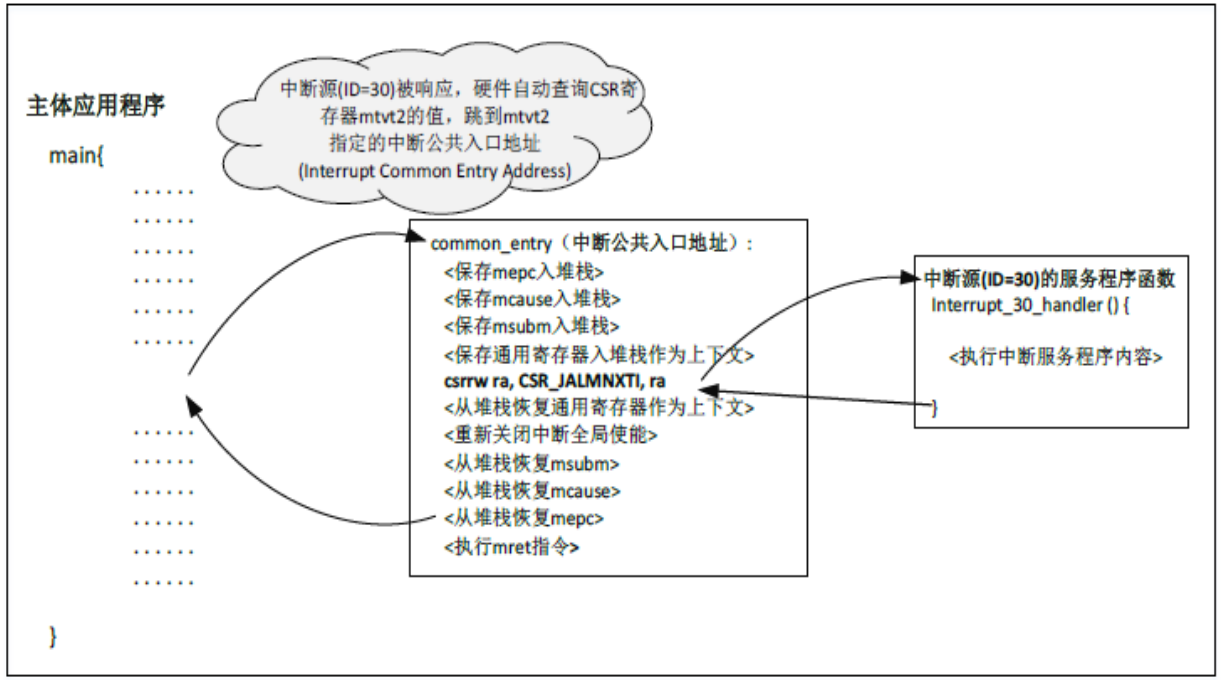
《Cortex™-M3 Technical Reference Manual》,Revision r2p0，ARM

《The RISC-V Instruction Set Manual, Volume I：Unprivileged ISA》， Ver. 20190608

异常响应过程



向量（M3，RV32）



非向量（RV32）

《Cortex™-M3 Technical Reference Manual》,Revision r2p0，ARM

《The RISC-V Instruction Set Manual, Volume I：Unprivileged ISA》， Ver. 20190608

RV32 教学-对比

指令

Operation	M3	RV32IMAC
Move	MOV, MVN	
Addition	ADD	ADD(I)
Subtract	SUB	SUB(I)
Multiply	MUL MLA	MUL MULHSU
Divide	SDIV UDIV	DIV DIVU REM(U)
Saturate	SSAT	
Compare	CMP	
Logical	AND EOR ORR BIC TST TEQ	AND(I) OR(I) XOR(I)
Shift	LSL LSR ASR	SLL SRL SRA

Operation	M3	RV32IMAC
Rotate	ROR RRX	
count	CLZ	
Load	LDR LDRH LDRB LDRD LDM	LB(U) LHW(U) LHW LUI
Store	STR STRH STRB STRD STM	SB SW SHW SLT(I)(U)
PUSH	PUSH	
POP	POP	
Semaphore	LDREX STREX CLREX	

Operation	M3	RV32IMAC
Branch	B BL BX BLX CBZ TB(B or H)	BEQ BNE BLT(U) BGE(U) JAL JALR
State Change	SVC CPSID CPSIE MRS MSR BKPT	ECALL CSRRC(I) CSRRS(I) CSRRW(I) EBREAK
Extend	SXTH SXTB UXTH UXTB	

Operation	M3	RV32IMAC
Bit Field	UBFX SBFX BFC BFI	
Reverse	REV REV16 REVSH RBIT	
Hint	SEV WFE WFI	MRET SRET WFI WFE

《Cortex™-M3 Technical Reference Manual》,Revision r2p0，ARM

《The RISC-V Instruction Set Manual, Volume I：Unprivileged ISA》， Ver. 20190608

调用规范

C函数	Cortex-M4汇编	RV32imac汇编

int add6(int p1, int p2, int p3, int p4, int p5, int p6)
 	
{
return(p1 + p2 + p3 + p4 + p5 + p6);
}
	.global add6
	.thumb
	.type add6, %function
	add6:
	add a0, a0, a1
	add a0, a0, a2
	add a0, a0, a3
	ldr a3, [sp]
	add a0, a0, a3
	ldr a3, [sp, #4]
	add a0, a0, 3
	bx lr
		.global add6
		.type add6, %function
		add6:
		add a0, a0, a1
		add a0, a0, a2
		add a0, a0, a3
		add a0, a0, a4
		add a0, a0, a5
		ret

RV32 教学-指令集模块

i vs im

C函数	RV32i	RV32im

int add4(int p1, int p2, int p3, int p4)
{
return (p1 * p2 + p3 / p4);
}
	add4:
	mv s1, a2
	mv s2, a3
	call __mulsi3
	mv s0, a0
	mv a1, s2
	mv a0, s1
	call __divsi3
	add a0, s0, a0
	jr ra
		add4:
		div a2, a2, a3
		mul a0, a0, a1
		add a0, a0, a2
		ret

编译结果对照

	RV32i	RV32im
得分（/MHZ）	1.33	2.43

CoreMark对照

im vs imc

目标文件	M4	RV32im	RV32imc
core_list_join.o	988	1884	1224
core_matrix.o	760	1356	868
core_state.o	622	1104	732
core_util.o	174	284	178

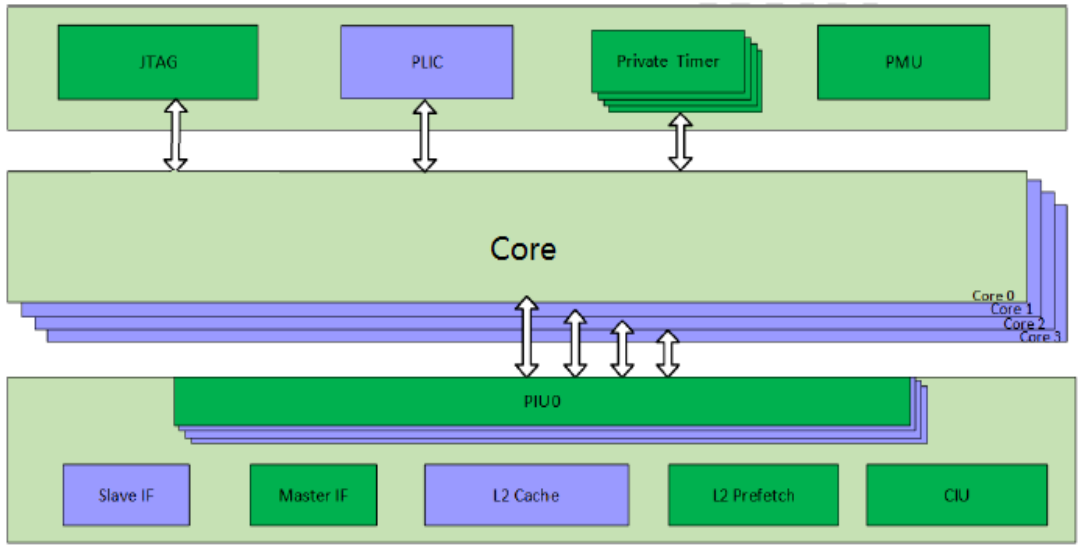
汇编结果对照

内容提要

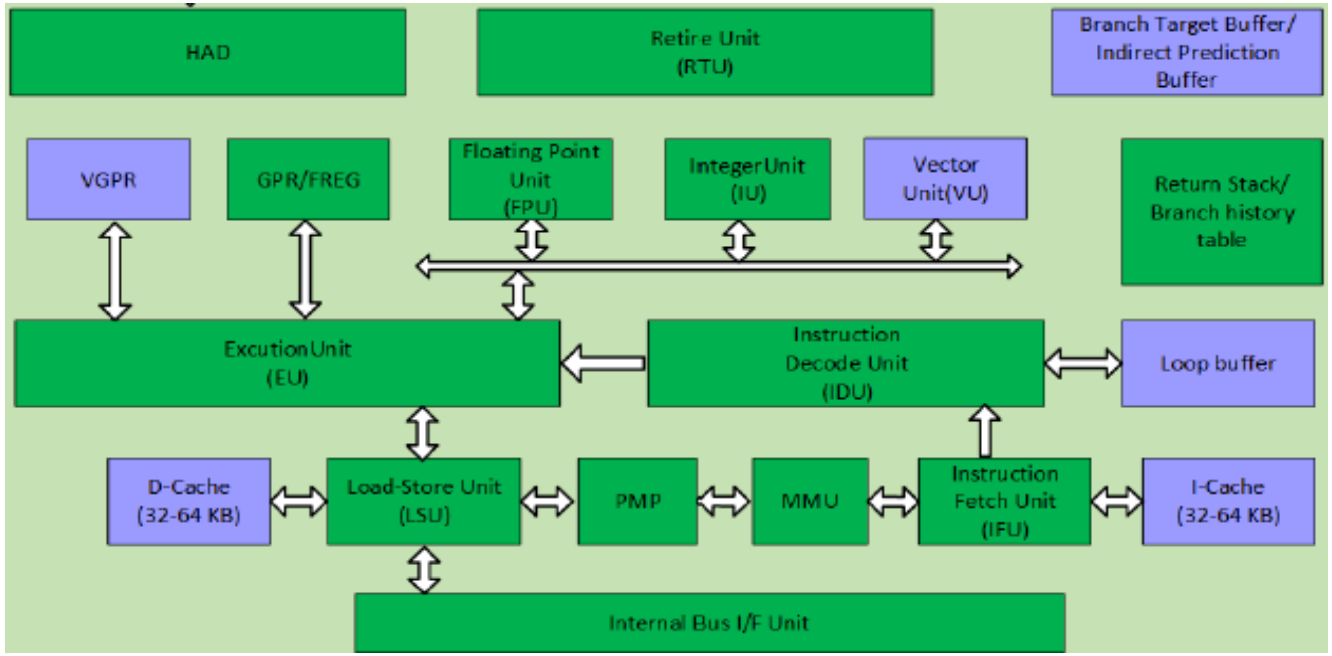
- 课程介绍
- RV32教学
- RV64教学
- 应用与实践
- 《深入理解RISC-V程序开发》（第二版）
- 小结

- C910 处理器核特点
  - 结构
  - 编程模式
- 应用系统
  - TH1520 SoC
  - 软件系统结构
  - 构建应用

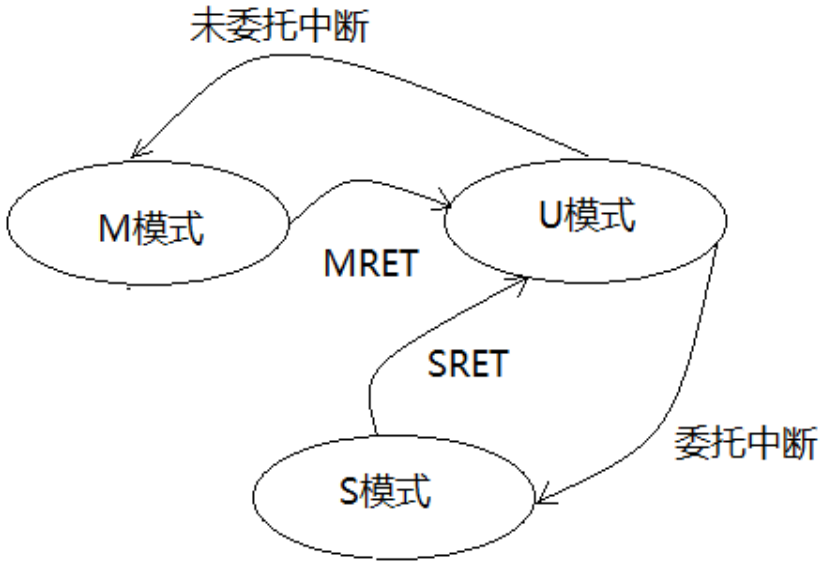
系统


JTAG
PLIC
Private Timer
PMU
Core
Core 0
Core 1
Core 2
Core 3
PIU0
Slave IF
Master IF
L2 Cache
L2 Prefetch
CIU

内核


HAD
Retire Unit
(RTU)
Branch Target Buffer/
Indirect Prediction
Buffer
VGPR
GPR/FREG
Floating Point
Unit
(FPU)
IntegerUnit
(IU)
Vector
Unit(VU)
Return Stack/
Branch history
table
ExcutionUnit
(EU)
Instruction
Decode Unit
(IDU)
Loop buffer
D-Cache
(32-64 KB)
Load-Store Unit
(LSU)
PMP
MMU
Instruction
Fetch Unit
(IFU)
I-Cache
(32-64 KB)
Internal Bus I/F Unit

编程模型（Program mode）



特权模式

……

A+7	A+6	A+5	A+4	A+3	A+2	A+1	A	
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	Double word at A
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	Word at A
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	Half word at A
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0	Byte at A

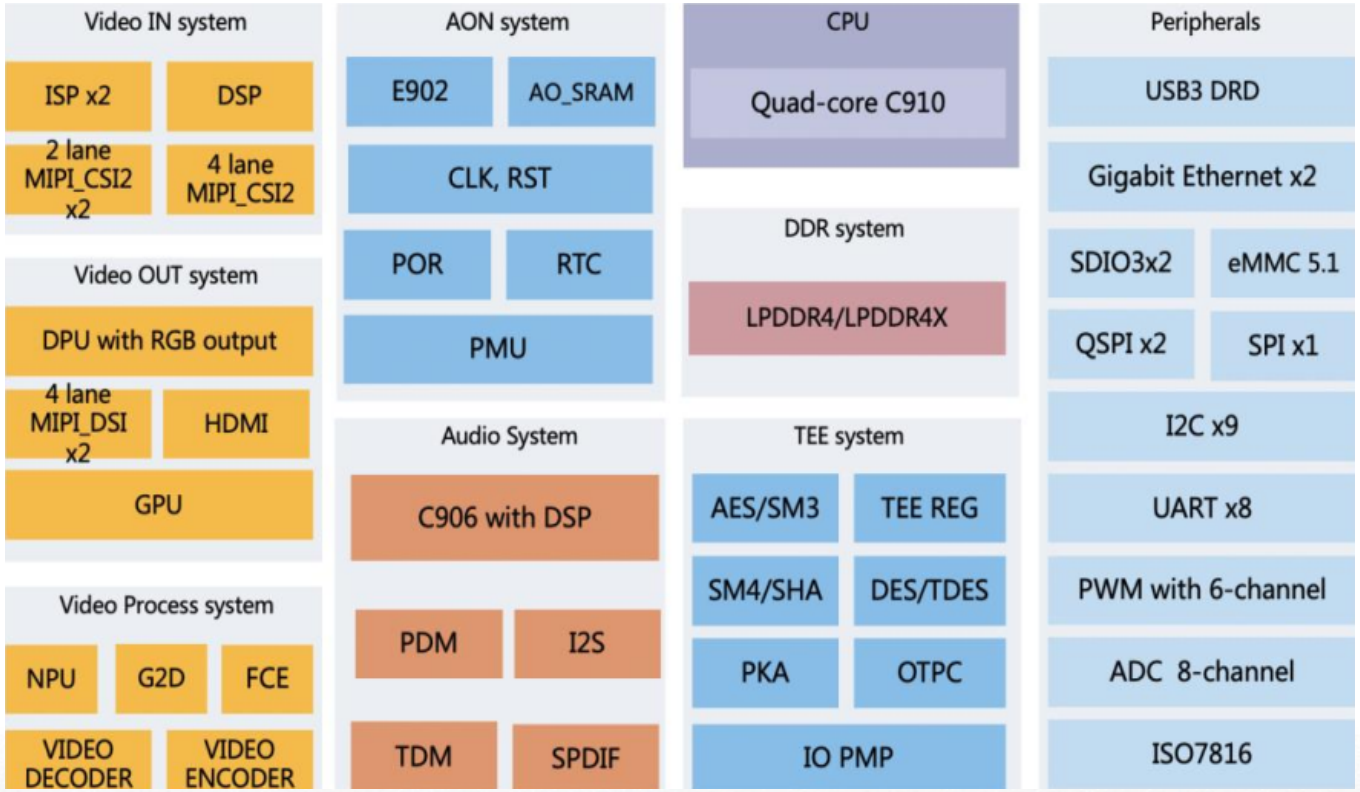
内存数据格式

```
fmv.h.x fd, rs1    // fd[15:0] <- rs1[15:0]; fd[63:16] <- 48'hffffffffffff
fmv.x.h rd,fs1      //tmp[15:0] <- fs1[15:0] ;rd <- sign_extend(tmp[15:0])
fmv.w.x fd, rs1     // fd[31:0] <- rs1[31:0]; fd[63:32] <- 32'hffffffff
fmv.x.w rd,fs1      //tmp[31:0] <- fs1[31:0] ;rd <- sign_extend(tmp[31:0])
fmv.d.x fd, rs1         // fd[63:0] <- rs1[63:0];
fmv.x.d rd,fs1      //rd <- fs1[63:0];
```

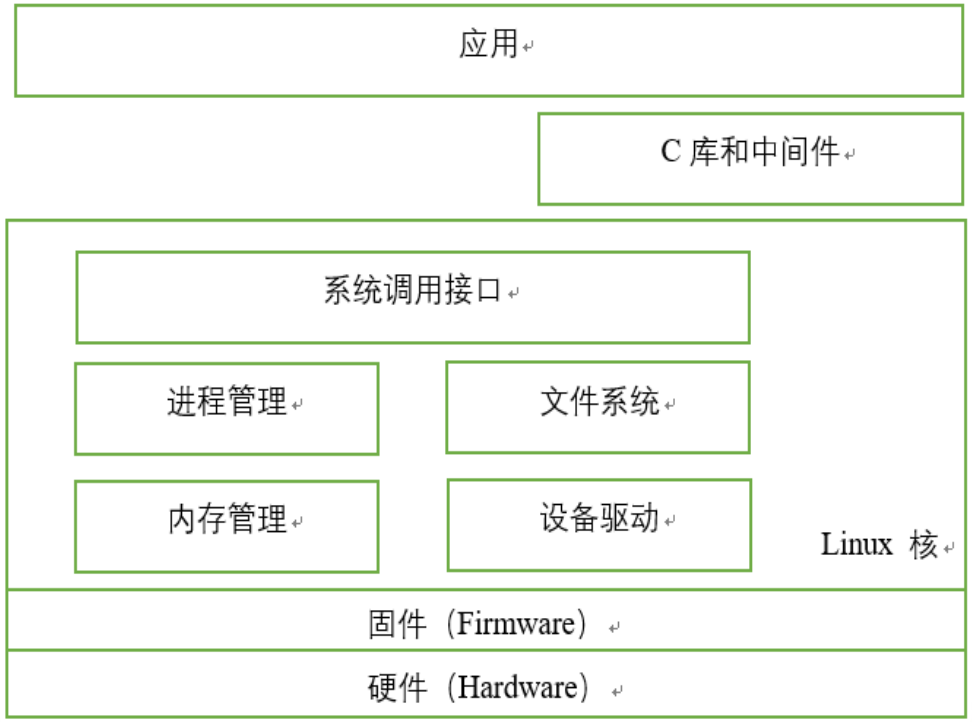
浮点和矢量计算

RV64 教学-应用系统

TH1520



软件系统



北京大学

构建应用

（1）配置

配置交叉工具链，以及应用软件依赖组件头文件与动态库。具体步骤如下：

1） 获取交叉工具链；

2） 获取发布镜像中的 rootfs ext4 镜像文件（如 rootfs.thead-image-xfce.ext4）

3） 配置 Linux 的 PATH 环境变量，加入交叉工具链的 bin 目录；最后，挂载 rootf 以获取头文件与动态库。

命令如下：

```
$ mkdir ~/thead-image-rootfs
$ sudo mount -t ext4 rootfs.thead-image-xfce.ext4 ~/thead-image-rootfs/
```

（2）编译

先在编译服务器上准备测试应用程序代码 myapp_gtk.c，源码如下：

```
$ mkdir ~/thead-image-rootfs
$ sudo mount -t ext4 rootfs.thead-image-xfce.ext4 ~/thead-image-rootfs/
$ riscv64-linux-gcc myapp_gtk.c -o myapp_gtk \
-I~/thead-image-rootfs/usr/include \
-I~/thead-image-rootfs/usr/include/gtk-3.0 \ -I~/thead-image-rootfs/usr/include/glib-2.0 \ -I~/thead-image-rootfs/usr/lib/glib-2.0/include \ -I~/thead-image-rootfs/usr/include/pango-1.0 \ -I~/thead-image-rootfs/usr/include/harfbuzz \ -I~/thead-image-rootfs/usr/include/cairo \ -I~/thead-image-rootfs/usr/include/gdk-pixbuf-2.0 \ -I~/thead-image-rootfs/usr/include/atk-1.0 \
-L~/thead-image-rootfs/usr/lib \
--no-sysroot-suffix --sysroot=~/thead-image-rootfs/ \ -lgobject-2.0 -lglib-2.0 -lgtk-3
```

(4) 部署

通过网络、U 盘、tf 存储卡拷贝等方法将应用程序 myapp_gtk 复制到设备中，然后运行。

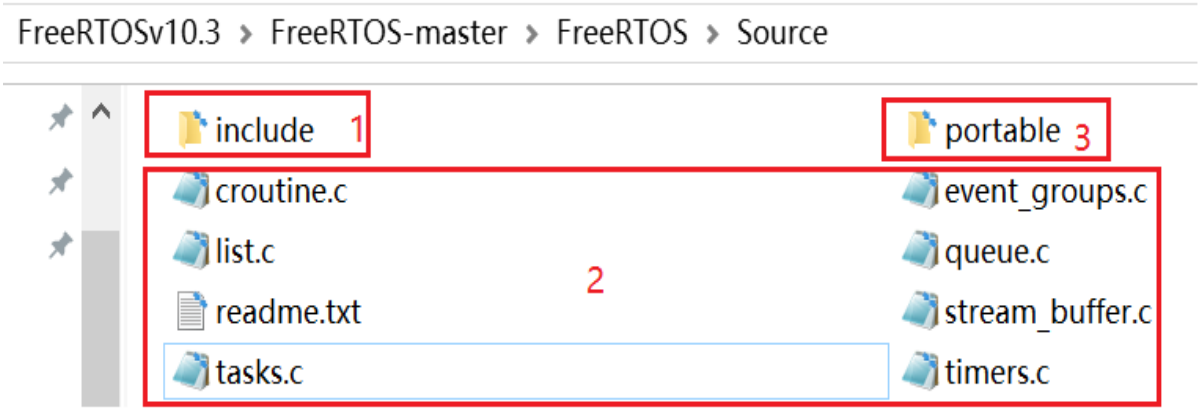
内容提要

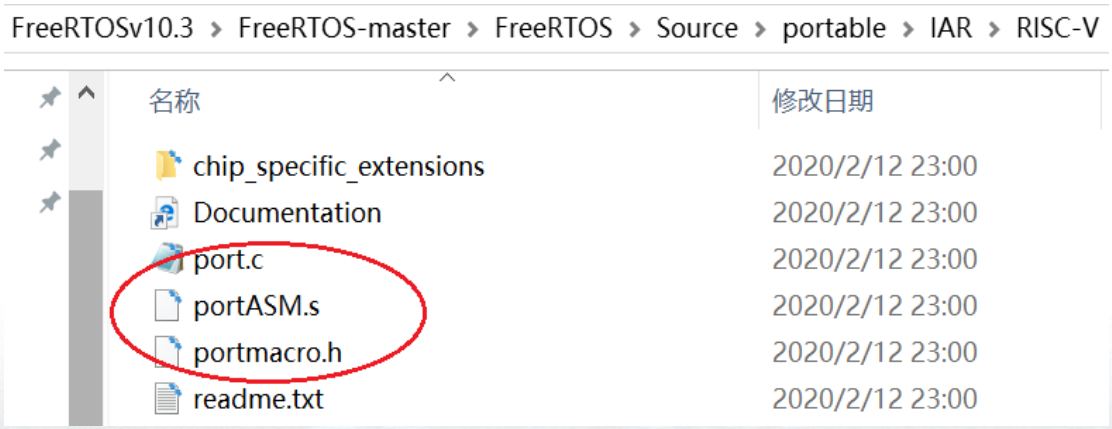
- 课程介绍
- RV32教学
- RV64教学
- 应用与实践
- 《深入理解RISC-V程序开发》（第二版）
- 小结

- 移植 FreeRTOS（GD32VF103）
- 姿态识别（TH1520）

任务内容

- 开启和关闭中断方式
- 进入和推出临界区的方式
- 产生周期性的中断，作为系统时钟节拍
- 任务的上下文切换



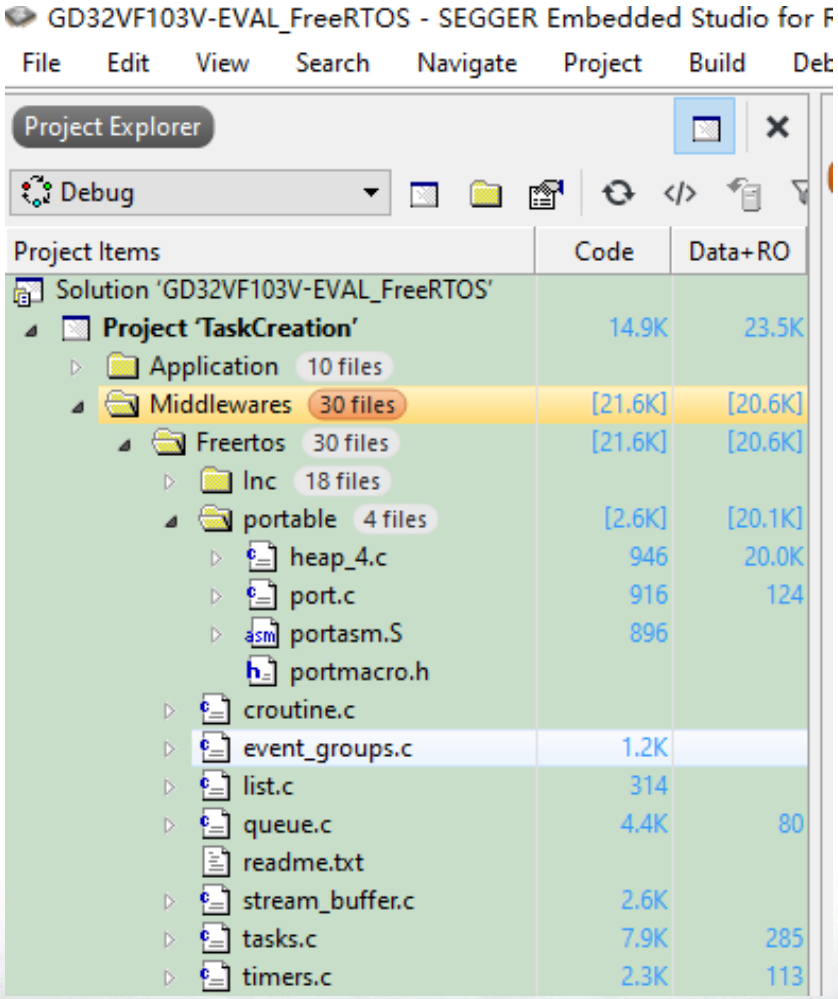


https://sourceforge.net/projects/freertos/files/FreeRTOS/

《深入理解RISC-V程序开发》，北航出版社

应用与实践-移植RTOS

工程



Port.c

进入临界区

```
void vPortEnterCritical( void ){
        #if USER_MODE_TASKS
                ECALL(IRQ_DISABLE);
        #else
                eclic_set_mth ((configMAX_SYSCALL_INTERRUPT_PRIORITY)<<4);
        #endif
        uxCriticalNesting++;}
```

退出临界区

```
void vPortExitCritical( void ){
        configASSERT( uxCriticalNesting );
        uxCriticalNesting--;
        if( uxCriticalNesting == 0 ){
                #if USER_MODE_TASKS
                        ECALL(IRQ_ENABLE);
                #else
                        eclic_set_mth (0);
                #endif}
        return; }
```

打开中断

```
void vPortClearInterruptMask(int int_mask){
        eclic_set_mth (int_mask);
}
```

屏蔽中断

```
int xPortSetInterruptMask(){
        int int_mask=0;
        int_mask=eclic_get_mth();
        eclic_set_mth ((configMAX_SYSCALL_INTERRUPT_PRIORITY)<<4);
        return int_mask;
}
```

上下文堆栈

```
StackType_t *pxPortInitialiseStack( StackType_t *pxTopOfStack, TaskFunction_t pxCode, void *pvParameters )
{       // Simulate the stack frame as it would be created by a context switch interrupt.
        register int *tp asm("x3");
        pxTopOfStack--;
        *pxTopOfStack = (portSTACK_TYPE)pxCode;  // Start address
        //set the initial mstatus value
        pxTopOfStack--;
        *pxTopOfStack = MSTATUS_INIT;
        pxTopOfStack -= 22;
        *pxTopOfStack = (portSTACK_TYPE)pvParameters;        /* Register a0 */
        pxTopOfStack -=9;
        *pxTopOfStack = (portSTACK_TYPE)prvTaskExitError; /* Register ra */
        pxTopOfStack--;
        return pxTopOfStack;
}
```

定时器中断服务

```
uint32_t vPortSysTickHandler(){
        static uint64_t then = 0;
        volatile uint64_t * mtime       = (uint64_t*) (TMR_CTRL_ADDR + TMR_MTIME);
        volatile uint64_t * mtimecmp    = (uint64_t*) (TMR_CTRL_ADDR + TMR_MTIMECMP);
        if(then != 0) { //next timer irq is 1 second from previous
                then += (configRTC_CLOCK_HZ / configTICK_RATE_HZ);
        } else{ //first time setting the timer
                uint64_t now = *mtime;
                then = now + (configRTC_CLOCK_HZ / configTICK_RATE_HZ);
        }
        *mtimecmp = then;
        /* Increment the RTOS tick. */
        if( xTaskIncrementTick() != pdFALSE )
        {
                portYIELD();
        }
}
```

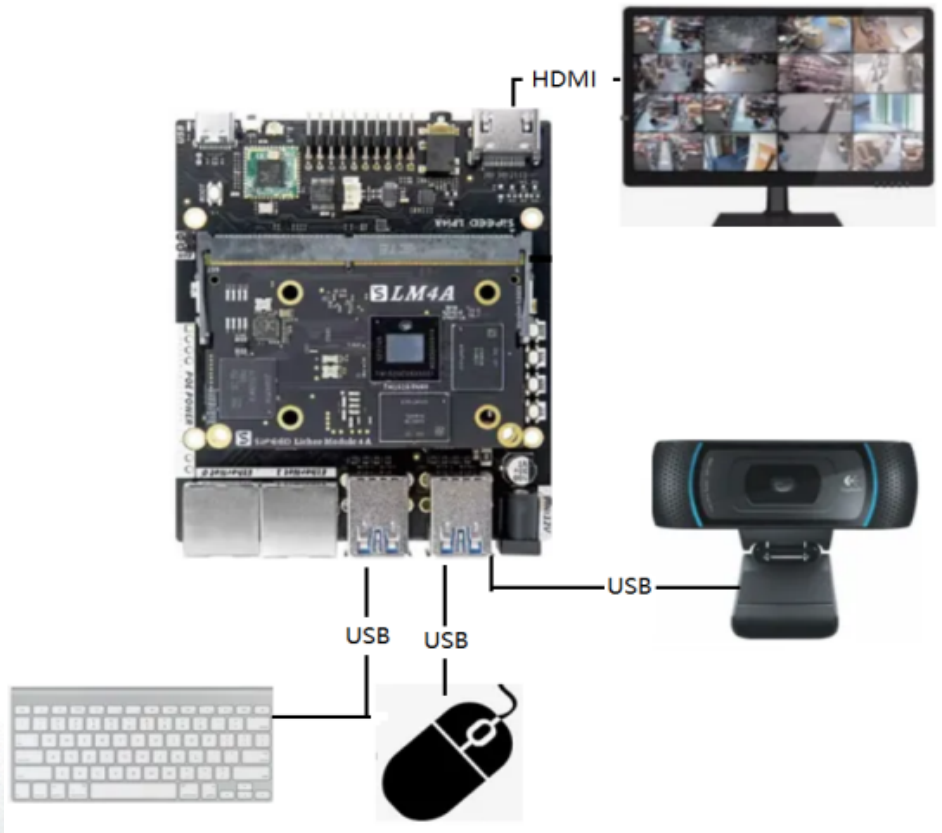
https://sourceforge.net/projects/freertos/files/FreeRTOS/

《深入理解RISC-V程序开发》，北航出版社

北京大学

应用与实践-姿态检测

系统



算法支持环境

Model Format	.onnx(PyTorch)	.pb(TensorFlow)	.caffemodel(Caffe)	···
HHB	Command line interface	Python API		
	Quantization Graph Optimization			
	Graph Codegen	ONNX Runtime	Host Simulate	Model Profile
	Target Deploy			
SHL	CSI-NN2 API			
	Heterogeneous Scheduler			
	CPU ASM OPT	NPU Driver		
Hardware	CPU	NPU		

北京大学

应用与实践-姿态检测

检测过程

串接模型过程可分为如下 6 步：

（1） 加载NPU的int16模型；

（2） 加载CPU的float16模型；

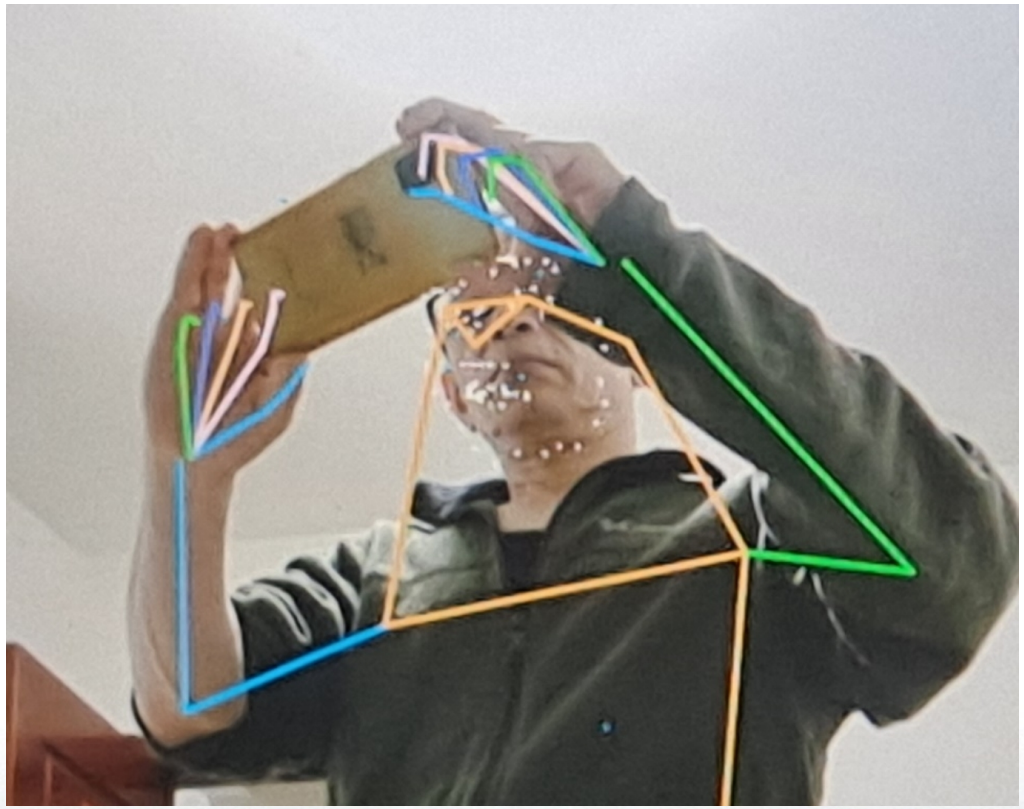
（3） 对输入预处理；

（4） 将输入提供给模型第一部分；

（5） 将第一部分的输出提供给模型的第二部分；

（6） 对第二部分的输出做后处理

效果

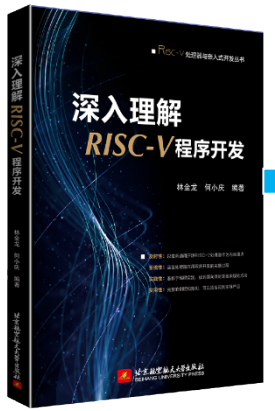


北京大学

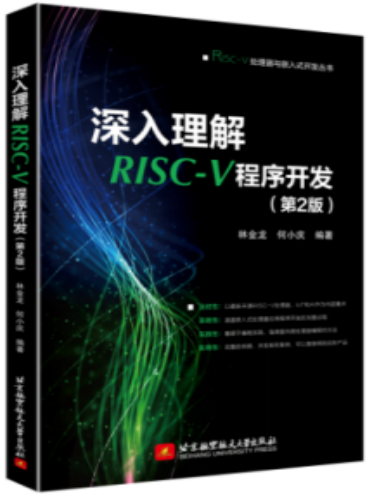
内容提要

- 课程介绍
- RV32教学
- RV64教学
- 应用与实践
- 《深入理解RISC-V程序开发》（第二版）
- 小结

《深入理解RISC-V程序开发（第二版）



2021年9月第一版



2025年8月第二版

- 补充和更新了4年来出现的新RISC-V处理器和开发工具
- 新增了RV64指令集阐述和特点介绍
- 介绍了嵌入式处理器TH1520 SoC 和软件环境
- 介绍了RT-Thread最新商业产品内容
- 新增了Zephyr RTOS 实例，TH1520 SoC荔枝派4A开发板姿态检测实例
- 介绍了RV64处理器人工智能应用开发过程

北京大学

小结

- 教学是推广RISC-V处理器，发展RISC-V生态的重要环节；
- 目前在《计算机体系结构》与《微处理器设计》课程中已广泛引入RISC-V处理器架构；
- 强化在处理器应用相关课程中引入RISC-V处理器，将直接和快捷地使更多未来的开发人员熟悉和使用RISC-V；
- 《深入理解RISC-V程序开发》第二版在8月与读者见面，我们计划与产业伙伴对线上讲座和课件进行更新；希望带动更多高校开设RISC-V 课程，吸引更多开发者学习RISC-V 开发知识。

嵌入式课件和实验
代码下载

谢谢，敬请指正！

欢迎参观C30 SEGGER/麦克泰技术 展区
- 图书和GD32 RISC-V开发板展示