玄铁最小系统RTOS SDK使用参考手册

版本	修订内容	作者	时间
v2.0	初始版本，配套玄铁RTOS SDK V2.X使用	尹桂才	2024/04/08

缩写/术语	定义
CSI	Chip System Interface。玄铁定义的一套嵌入式软件接口标准，该接口包含了CPU 核、芯片驱动程序、DSP 指令等部分的接口抽象。
CSI-Core	针对CPU 硬件的CSI 接口
CSI-Driver	针对外围驱动的CSI 接口
CDS	玄铁开发的基于Eclipse 的IDE
CDK	玄铁开发的适用于物联网和MCU 领域的IDE
yoctools	玄铁开发的一套基于scons的组件化编译构建工具
smartl	玄铁内部自研FPGA平台，针对E9xx系列。玄铁QEMU模拟器中有模拟对应的machine
xiaohui	玄铁内部自研FPGA平台，针对C/R9xx系列。玄铁QEMU模拟器中有模拟对应的machine

1. 介绍

1.1. 概述

玄铁最小系统RTOS SDK 是玄铁处理器配套的软件开发工具包，软件遵循 CSI 接口规范。用户可通过该 SDK快速对玄铁处理器进行测试与评估，也可参考 SDK 中集成的各种常用组件以及示例程序进行应用开发，快速形成产品方案。该文档可搭配RTOS-SDK-CPUTYPE-V2.x.x.zip版本的SDK(最小系统RTOS SDK发布形态有两种，具体请参考第2节说明)使用。配套的SDK兼容玄铁所有C9xx/R9xx/E9xx处理器型号，可供OS开发工程师、应用开发工程师、SOC设计&测试开发人员参考使用。

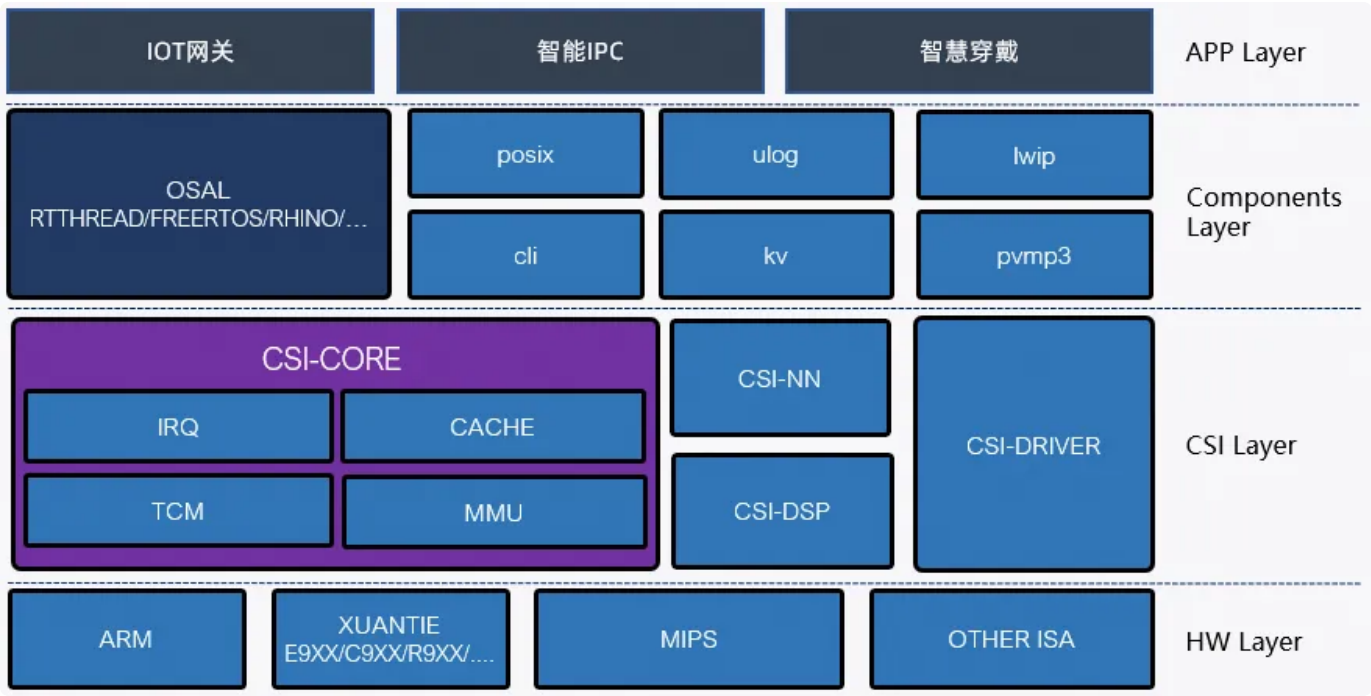
SDK中当前提供了freertos(仅单核)和rtthread(支持SMP)操作系统的移植适配，并可基于玄铁QEMU模拟器运行。其中对于E9xx系列处理器，仅支持smartl平台，对于C/R9xx系列处理器，仅支持xiaohui平台。SDK中提供了面向三种级别(裸系统、原生RTOS、基于OSAL封装)的使用示例，用户可根据不同的需求参考使用。示例大致提供了三种编译方式，分别可基于Linux命令行、CDK、CDS构建编译运行。由于每种构建编译支持的特性差异，部分示例稍有区别，具体请查看第3节描述。其中，一、所有示例在Linux命令行/CDS下均可编译，二、CDK仅支持玄铁E9xx系列编译，三、部分示例由于跟硬件强相关，QEMU中未模拟，所以不支持模拟器运行。

1.2. SDK架构

玄铁RTOS SDK V2.X较1.X版本整体设计有较大不同，一是将之前E9xx和C9xx的RTOS SDK整合成一个仓库统一起来，二是从CSI1.0切换到CSI2.0接口标准，三是采用组件化的思想(基于yoctools构建)并根据玄铁处理器不同客户的特点，大致分为三个级别提供相关的组件，如下：

级别	说明	客户群体
baremetal level	面向裸程序场景，提供玄铁cpu中断、cache、tcm、ecc等core相关用法。应用通过简单轮询即可，存在少量外设，无OS任务调度需求等。一般ram < 50KB	直接基于RTOS SDK开发或根据core相关示例移植到厂商自有软件框架中
mcu level	面向mcu firmware级别，对RTOS有需求(倾向直接调用原生OS接口)、应用相对不复杂、外设较少、轻量文件系统(如KV)的场景。一般ram < 1MB，无C++、SD卡、LWIP等依赖	直接基于RTOS SDK开发或根据指定RTOS相关示例移植到厂商自有软件框架中
soc level	面向解决方案级别，应用较复杂、外设较丰富的场景，如智能音箱、猫眼门锁、IPC等。一般上层使用posix接口(不直接调用原生OS接口)，有C++、SD卡、LWIP、蓝牙协议栈、VFS等需求，ram > 1MB	直接基于RTOS SDK开发，厂商RTOS软件框架无积累或不完善

SDK中提供了面向三种级别的使用示例，包括中断、低功耗、基本外设驱动、benchmark、原生内核(freertos & rtthread)核心接口、内核无关(采用osal封装)核心接口等。用户可直接使用对应的示例对玄铁处理器进行评测，也可根据具体产品的特点和需求直接基于不同级别的示例移植开发对应的解决方案。配套的SDK架构图如下：



整个SDK大致分为四层，如下：

- 硬件层：可通过CSI层支持基于玄铁各系列处理器的SOC。用户也可根据CSI接口规范支持其他处理器
- CSI接口层：针对嵌入式系统，定义了CPU内核移植接口、外围设备操作接口统一软件接口规范。其通过消除不同芯片的差异，可简化软件的使用及提高软件的移植性。
- 组件层：提供了诸如日志、命令行、文件系统、网络协议栈、编解码等各种类型的组件。官方ZIP包中仅提供了最小依赖组件，用户可参考第5节中相关描述下载其他所需要的组件
- 解决方案层：官方ZIP包中仅提供了玄铁处理器的基本示例，如中断、低功耗、RTOS核心接口设施使用等功能。更丰富的解决方案如面向网关接入、智能音箱、猫眼门锁等请参考第5节中相关描述下载

其中CSI层主要内容包括：

- CSI-CORE：定义了CPU和相关紧耦合外设的接口规范
- CSI-DRIVER：定义了常用的驱动的接口规范

1.3. 目录结构

玄铁处理器RTOS SDK代码目录结构如下：

Bash

```
.
├── boards
│   └── board_riscv_dummy
├── components
│   ├── aos
│   ├── chip_riscv_dummy
│   ├── cli
│   ├── csi
│   ├── debug
│   ├── devfs
│   ├── devices
│   ├── drivers
│   ├── freertos
│   ├── kv
│   ├── libc_bare
│   ├── libc_freertos
│   ├── libc_rtthread
│   ├── libc_stub
│   ├── partition
│   ├── posix
│   ├── rtthread
│   ├── sdk_chip_riscv_dummy
│   ├── ulog
│   ├── uservice
│   ├── vfs
│   ├── xuantie_cpu_sdk
│   └── RTOS SDK_base
└── solutions
    ├── bare_core_dsp
    ├── bare_core_ecc
    ├── bare_core_lockup
    ├── bare_coremark
    ├── bare_core_matrix
    ├── bare_core_nmi
    ├── bare_core_tcm
    ├── bare_core_vector
    ├── bare_core_vic
    ├── bare_core_wfe
    ├── bare_core_wfi
    ├── bare_cpp_demo
    ├── bare_dhrystone
    ├── bare_drv_timer
    ├── bare_drv_uart
    ├── bare_helloworld
```

```
45      ├── bare_linpack_dp
46      ├── bare_linpack_sp
47      ├── bare_whetstone
48      ├── mcu_freertos_cpp
49      ├── mcu_freertos_event
50      ├── mcu_freertos_helloworld
51      ├── mcu_freertos_message_q
52      ├── mcu_freertos_mutex
53      ├── mcu_freertos_sem
54      ├── mcu_freertos_task
55      ├── mcu_freertos_time
56      ├── mcu_freertos_timer
57      ├── mcu_rtthread_cpp
58      ├── mcu_rtthread_event
59      ├── mcu_rtthread_helloworld
60      ├── mcu_rtthread_message_q
61      ├── mcu_rtthread_mutex
62      ├── mcu_rtthread_sem
63      ├── mcu_rtthread_task
64      ├── mcu_rtthread_time
65      ├── mcu_rtthread_timer
66      ├── soc_core_dsp
67      ├── soc_core_matrix
68      ├── soc_core_vector
69      ├── soc_helloworld
70      ├── soc_kernel_event
71      ├── soc_kernel_message_q
72      ├── soc_kernel_mutex
73      ├── soc_kernel_sem
74      ├── soc_kernel_task
75      ├── soc_kernel_time
76      ├── soc_kernel_timer
77      └── soc_smp_demo
```

其中，主要目录说明如下表所示：

目录名	说明
boards	板级相关代码均存放在此目录中
boards/board_riscv_dummy	玄铁板级dummy组件。用户若有新的板级需求，可基于此拷贝
components	内核、chip、c库、posix、osal等基础组件均存放在此目录中
components/aos	操作系统抽象层(osal)组件
components/csi	CSI接口定义

components/chip_riscv_dummy	基于玄铁处理器的chip dummy组件。若开发新的soc，可基于此拷贝
components/cli	基于osal接口实现的命令行组件
components/devices	rvm-hal设备驱动框架
components/freertos	FreeRTOS内核及玄铁arch相关代码
components/rtthread	RTThread内核及玄铁arch相关代码
components/libc_stub	基础c库，部分接口如printf基于osal接口对接
solutions	解决方案层组件，提供了面向裸系统、原生RTOS等三个层次的应用示例
solutions/bare_xxx	面向裸系统的基础示例，包含中断、tcm、c++、性能测试等
solutions/mcu_freertos_xxx	基于freertos原生基础内核设施接口的示例，如锁、信号量、消息队列
solutions/mcu_rtthread_xxx	基于rtthread原生基础内核设施接口的示例，如事件、时间、消息队列
solutions/soc_xxx	基于osal封装的内核设施接口的示例，如任务、定时器、SMP等

2. 具体使用

2.1. 环境依赖

用户可基于Windows环境下的CDK/CDS或Linux环境(推荐)下开发。若基于Windows开发时，请使用最新的工具版本开发。具体下载地址请参考最后一节内容。若基于Linux环境下使用，推荐依赖的环境如下：

工具/环境	版本
UBUNTU	20.04
python	3.6+
pip	21.3.1+

yoctools构建工具	2.0.76+
Xuantie-900-gcc-elf-newlib-x86_64编译工具链	2.10.0+
玄铁模拟器	4.2.9+

2.2. 玄铁工具下载&安装

由于玄铁处理器实现与RV官方标准存在部分差异(如vector指令、玄铁其他扩展指令等)，因此相关示例需要搭配玄铁对应的工具编译运行等。

2.2.1. 工具链

工具链的下载请参考第6节相关说明。安装参考如下：

```Bash
#下载后解压到/opt目录
sudo tar zxvf Xuantie-900-gcc-elf-newlib-x86_64-V2.8.0-20231018.tar.gz -C /opt/.
#配置环境变量，~/.bashrc文件尾部添加下行内容并生效
export PATH="/opt/Xuantie-900-gcc-elf-newlib-x86_64-V2.8.0/bin:$PATH"
source ~/.bashrc
#查看对应的版本号
riscv64-unknown-elf-gcc -v
```

2.2.2. 玄铁模拟器

玄铁模拟器的下载请参考第6节说明。安装参考如下：

```bash
#安装到/opt目录下
tar zxvf Xuantie-qemu-x86_64-Ubuntu-22.04-V4.2.3-B20231025-1012.tar.gz
sudo mv install /opt/Xuantie-qemu-x86_64-Ubuntu-22.04-V4.2.3-B20231025-1012
#配置环境变量，~/.bashrc文件尾部添加下行内容并生效
export PATH="/opt/Xuantie-qemu-x86_64-Ubuntu-22.04-V4.2.3-B20231025-1012/bin:$PATH"
source ~/.bashrc
#查看对应的版本号
qemu-system-riscv64 --version
```

2.2.3. yoctools构建工具

玄铁RTOS SDK在Linux环境下是基于自研的yoctools工具编译构建。该工具通过pip下载，可基于python2/3运行(推荐python3.6及以上版本)，命令参考如下：

```bash
#若之前未安装yoctools
sudo pip install yoctools
#若之前已安装yoctools，可升级对应包
sudo pip install -U yoctools
```

安装成功后，可通过如下命令查看版本号：

```bash
yingc@docker-ubuntu18:~/work/git/develop/tmp$ yoc --version
2.0.76
```

注意：

- 需要保证python和pip版本一致，如均是python2 or python3版本。否则可能出现安装异常
- 若下载不下来，可能是外网访问被禁。此时可通过阿里云或中科大pip等代理下载
- Linux环境下首次编译时，若之前未安装过玄铁工具链，yoctools会自动下载(建议手动到玄铁官方网站下载最新版本工具链并安装)。

2.3. 代码下载

当前玄铁RTOS SDK主要提供了两种代码下载方式，分别为官方指定CPU型号ZIP压缩包下载(推荐)和通过yoc(基于Linux环境，依赖yoctools)命令下载。两种方式下载的代码有些许不同，如下：

- 官方ZIP压缩包是针对指定玄铁处理器型号的，可直接通过make命令构建(也可通过do_build.sh脚本)
- yoc命令下载的代码是默认的玄铁处理器型号，用户可通过do_build.sh脚本加上相关参数构建。

2.3.1. 官方ZIP压缩包下载

请通过玄铁处理器官方网站下载指定CPU型号SDK，链接地址如下：

https://www.xrvm.cn/community/download?id=4222755144183386112

下载解压后，若基于Linux环境首次编译构建，请在对应解压目录下执行yoc init命令并安装最新版本工具链。

2.3.2. yoc命令下载

```Bash
#初始化工作空间(仅执行一次)
yoc init
#下载玄铁RTOS SDK(可通过https://gitee.com/yocop站点查看对应组件的分支)
yoc install xuantie_cpu_sdk -b master
```

2.4. 编译&运行

示例位于solutions目录下，提供了三种编译方式，分别可基于Linux命令行、CDK、CDS构建编译运行。由于每种构建编译支持的特性差异，部分示例稍有差别，具体请查看第3节描述。

所有编译出来的elf可基于玄铁自研FPGA平台(E9xx仅smartl平台，C/R9xx仅xiaohui平台)上运行。由于成本等原因，用户无法直接拿到对应的FPGA平台，此时可通过玄铁模拟器运行(仅部分示例支持)。若用户需要基于内部的FPGA平台使用这些示例，请参考第4节中的内容参考移植。

2.4.1. 命令行编译&运行

以solutions/bare_helloworld最小工程为例，编译命令如下：

```Bash
#以玄铁c907fdvm处理器为例(仅xiaohui平台)
./do_build.sh c907fdvm xiaohui
#以玄铁e907fdp处理器为例(仅smartl平台)
./do_build.sh e907fdp smartl
#clean命令
make clean
```

编译后，通常情况下同一个elf(除非某些示例额外说明)可通过玄铁模拟器或基于对应FPGA硬件平台运行。

注意：

若通过官方ZIP包下载解压后，需要在解压根目录中执行yoc init命令。编译某示例时可直接执行make命令

2.4.1.1. 基于玄铁QEMU运行

玄铁模拟器的下载请参考第6节相关说明。安装后，使用相关命令参考如下：

```Bash
#c907fdvm(单核)
qemu-system-riscv64 -machine xiaohui -nographic -kernel yoc.elf -cpu c907fdvm

#c907fdvm(双核)
qemu-system-riscv64 -machine xiaohui -smp cpus=2 -nographic -kernel yoc.elf -cpu c907fdvm

#c907fdvm-rv32(单核, C907 RV32模式)
qemu-system-riscv32 -machine xiaohui -nographic -kernel yoc.elf -cpu c907fdvm-rv32

#e907fdp
```

2.4.1.2. 基于玄铁FPGA硬件平台运行

基于FPGA运行时，需要提前安装好玄铁DebugServer(jtag)工具，软件下载请参考第6节相关内容。

2.4.1.2.1. DebugServer连接(jtag)

当bit成功下载到fpga平台后，此时可通过对应工具连接(以Linux系统为例)，连接命令参考如下：

Bash

```
jenkins@30.120.234.10 :~$  DebugServerConsole -prereset
+---                                                    ---+
|  T-Head Debugger Server (Build: Jan 28 2024, Linux)      |
   User   Layer Version : 5.17.22
   Target Layer version : 2.0
|  Copyright (C) 2023 T-HEAD Semiconductor Co.,Ltd.        |
+---                                                    ---+
T-HEAD: CKLink_Lite_V2, App_ver 2.37, Bit_ver null, Clock 2526.316KHz,
       5-wire, With DDC, Cache Flush On, SN CKLink_Lite_V2-T00000003636503
B6537894.
WARNING: executing abscmd 0x320cc1(read reg xrlenb) gets ABSTRACTCS.cmder
r 3(exception: an exception occurred while executing the ABSCMD sequenc
e). The result of the operation will not be trusted.
WARNING: Matrix module is found in CPUID, but failed to get XRLENB, defaul
t 16.
ERROR: CPU_1: Fail to enter debug mode.
WARNING: CPU_1: DMSTATUS is 0x4030a2, interpret as:
        NdmResetPending: 0, StickyUnavail: 0,   ImpEbreak: 1,       AllHav
eReset: 0
        AnyHaveReset: 0     AllResumeAck: 0,    AnyResumeAck: 0,    AllNon
exitent: 0
        AnyNonexistent: 0,  AllUnavail: 1,      AnyUnavail: 1,      AllRun
ning: 0
        AnyRunning: 0,      AllHalted: 0,       AnyHalted: 0,       Authen
ticated: 1
        AuthBusy: 0,        Hasresethaltreq: 1, Confstrptrvalid: 0, Versio
n: 2.
ERROR: CPU_2: Fail to enter debug mode.
WARNING: CPU_2: DMSTATUS is 0x4030a2, interpret as:
        NdmResetPending: 0, StickyUnavail: 0,   ImpEbreak: 1,       AllHav
eReset: 0
        AnyHaveReset: 0     AllResumeAck: 0,    AnyResumeAck: 0,    AllNon
exitent: 0
        AnyNonexistent: 0,  AllUnavail: 1,      AnyUnavail: 1,      AllRun
ning: 0
        AnyRunning: 0,      AllHalted: 0,       AnyHalted: 0,       Authen
ticated: 1
        AuthBusy: 0,        Hasresethaltreq: 1, Confstrptrvalid: 0, Versio
n: 2.
ERROR: CPU_3: Fail to enter debug mode.
WARNING: CPU_3: DMSTATUS is 0x4030a2, interpret as:
        NdmResetPending: 0, StickyUnavail: 0,   ImpEbreak: 1,       AllHav
eReset: 0

```

```
        AnyHaveReset: 0     AllResumeAck: 0,    AnyResumeAck: 0,    AllNon
exitent: 0
        AnyNonexistent: 0,  AllUnavail: 1,      AnyUnavail: 1,      AllRun
ning: 0
        AnyRunning: 0,      AllHalted: 0,       AnyHalted: 0,       Authen
ticated: 1
        AuthBusy: 0,        Hasresethaltreq: 1, Confstrptrvalid: 0, Versio
n: 2.
+--  Debug Arch is RVDM.  --+
+--  CPU 0  --+
RISCV CPU Info:
        WORD[0]: 0x09181b0e
        WORD[1]: 0x1000e000
        WORD[2]: 0x20000000
        WORD[3]: 0x30030066
        WORD[4]: 0x46590000
        WORD[5]: 0x50000000
        WORD[6]: 0x68000900
        MISA   : 0x8000000000b4112d
        MHARTID: 0x0
Target Chip Info:
        CPU Type is C907FDVM-rv64, Endian=Little, with TEE, Vlen=256, Mlen
b=128(Rlenb=16), Version is R0S0P14.
        DCache size is 32K, 4-Way Set Associative, Line Size is 64Bytes, w
ith ECC.
        ICache size is 32K, 4-Way Set Associative, Line Size is 64Bytes, w
ith ECC.
        MMU has 512 JTLB items.
        HWBKPT number is 3, HWWP number is 3.
        MISA: (RV64IMAFDCVX, Imp M-mode, S-mode, U-mode)

GDB connection command for CPUs(CPU0):
        target remote 127.0.0.1:1025
        target remote 30.120.234.10:1025

****************  DebuggerServer Commands List **************
help/h
        Show help informations.
************************************************************
```

当jtag连接成功后会出现target remote yourip:1025打印信息(注意：当仅有一个核释放时只会出现一个CPU0显示信息)。

注意：DebugServerConsole 连接命令对某些特殊用途的bit可能略有差异。若连接失败时，请咨询玄铁相关技术工程师

2.4.1.2.2. gdb下载程序&运行

在通过gdb下载程序到fpga之前，需要根据jtag连接成功后的打印信息，修改boards/board_riscv_dummy/script/xiaohui或者boards/board_riscv_dummy/script/smartl中对应cpu型号的gdbinit.xxx文件中target连接地址(其他寄存器的初始配置参考该gdbinit脚本，若有特殊需要可咨询对应的技术工程师修改)。

以boards/board_riscv_dummy/script/xiaohui/gdbinit.c907fdvm为例，参考如下：

Bash

```
target remote 30.120.234.10:1025

set confirm off
set height 0

reset
set $msmpr = 0x1



# Invalidate & Clear IBP BTB BHT ICache & DCache
set $mcor = 0x70013

# Enable L2 Cache
set $mccr2 = 0xa042000a

# Enable L1 Cache
set $mhcr = 0x10011bf

# Disable L1 & L2 Cache for debug
#set $mhcr = 0x108
#monitor set $mhint4=0x1000000

# Enable CPU Features
set $mxstatus = 0x638001
set $mhint = 0x3a1aa10c


##set $opensbi_addr = 0x00000000
#set $opensbi_addr = 0x80000000
#set $vmlinux_addr = $opensbi_addr + 0x00200000
#set $rootfs_addr  = $opensbi_addr + 0x04000000
#set $dtb_addr  = $rootfs_addr - 0x00100000
#set $dyn_info_addr  = $rootfs_addr - 0x40
#
## Load rootfs & kernel
#restore ../rootfs.cpio.gz binary $rootfs_addr
#restore ../Image binary $vmlinux_addr
#
## Load dtb
#restore hw.dtb binary $dtb_addr
#set $a1 = $dtb_addr
#set $a2 = $dyn_info_addr
#
```

```
#set *(unsigned long *)($dyn_info_addr) = 0x4942534f
#set *(unsigned long *)($dyn_info_addr + 8) = 0x1
#set *(unsigned long *)($dyn_info_addr + 16) = $vmlinux_addr
#set *(unsigned long *)($dyn_info_addr + 24) = 0x1
#set *(unsigned long *)($dyn_info_addr + 32) = 0x0
#set *(unsigned long *)($dyn_info_addr + 48) = 0x0
#
## Load opensbi
#restore ../fw_dynamic.bin binary $opensbi_addr

set *0x18030028=0x0
set *0x1803002c=0x0
set *0x18030030=0x0
set *0x18030034=0x0
set *0x18030038=0x0
set *0x1803003c=0x0
set *0x18030040=0x0
set *0x18030044=0x0

#file  ../vmlinux
#source ../gdbmacros.txt
lo
```

gdb下载运行，参考如下：

```
yingc@docker-ubuntu18:~/work/git/develop/xuantie/solutions/bare_helloworld
$ riscv64-unknown-elf-gdb yoc.elf -x ../../boards/board_riscv_dummy/scrip
t/xiaohui/gdbinit.c907fdvm
GNU gdb (Xuantie-900 elf newlib gcc Toolchain V2.10.0-rc5 B-20240319) 10.
0.50.20200724-git
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.ht
ml>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-un
known-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from yoc.elf...
0x0000000000000000 in ?? ()
0x0000000000000000 in ?? ()
Loading section .text, size 0x3ff8 lma 0x50000000
        section progress: 100.0%, total progress: 94.35%
Loading section .rodata, size 0x3d0 lma 0x50003ff8
        section progress: 100.0%, total progress: 99.98%
Loading section .data, size 0x4 lma 0x500043c8
        section progress: 100.0%, total progress: 100.00%
Start address 0x0000000050000034, load size 17356
Transfer rate: 136 KB/sec, 2479 bytes/write.
(gdb) c
Continuing.

```

注意：

- 根据DebugServer连接时出现的信息修改gdbinit脚本中的jtag连接命令
- gdbinit脚本请使用boards/board_riscv_dummy/script对应fpga平台中指定cpu型号默认配置
- 为清除上次执行程序时的潜在影响，每次下载程序运行前请重新烧写bit再通过DebugServerConsole –prereset命令(linux下)建立jtag连接

2.4.2. CDS编译&运行

所有示例均可基于CDS编译。CDS中默认集成了玄铁QEMU工具(windows版本)，示例编译后可通过简单配置即可基于QEMU运行(部分示例不支持QEMU运行，具体可参考第3节查看哪些示例支持)。

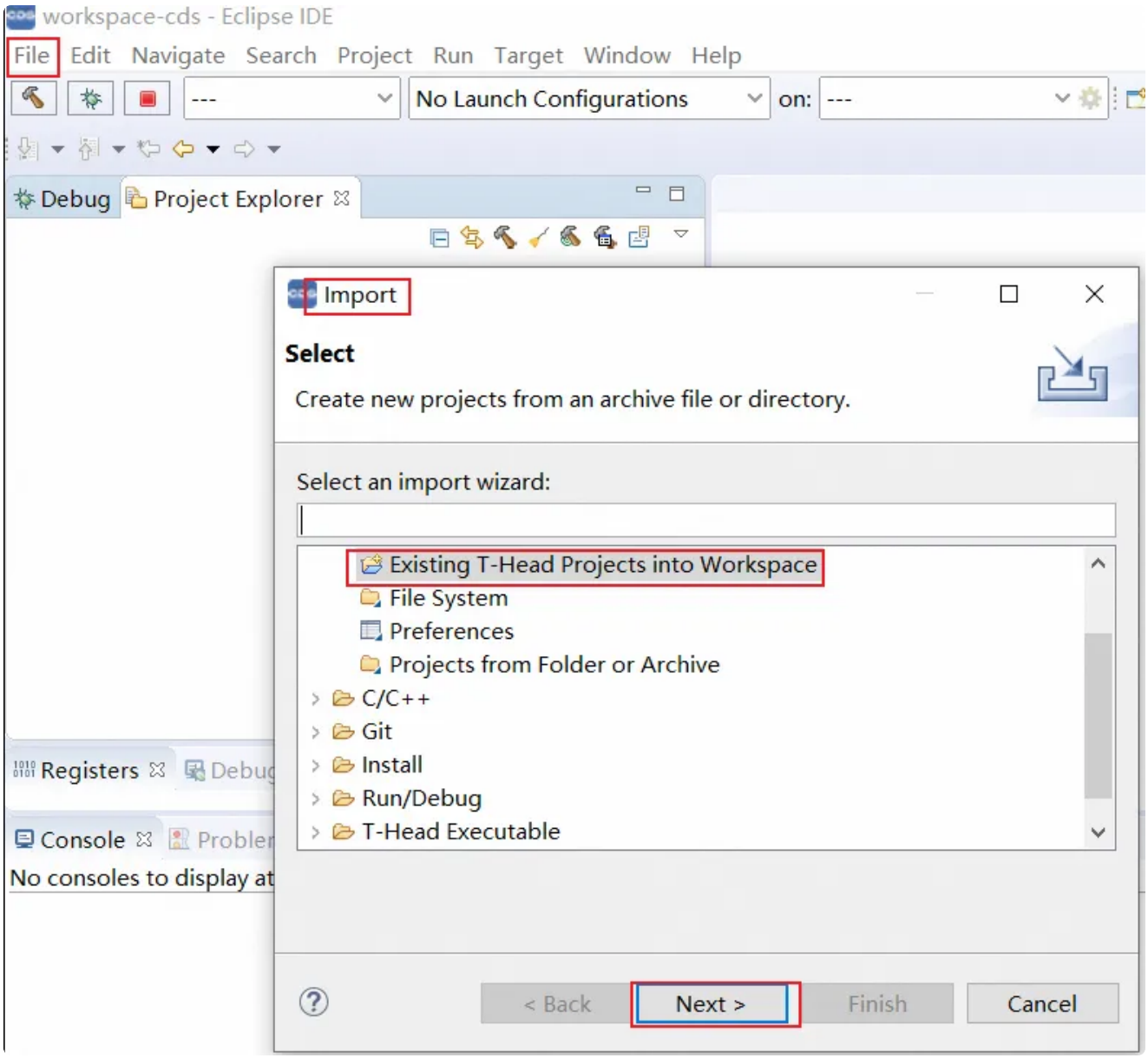
通过玄铁官方网站下载的ZIP形态SDK包默认在某示例文件夹下生成了对应玄铁CPU型号及默认RTOS类型的CDS工程，用户可参考下述小节内容编译运行。通过yoc命令下载的SDK示例中没有生成CDS工程，用户可参考以下命令基于Linux环境下生成：

```bash
#对于bare_xxx或mcu_xxx示例，在对应demo目录下输入以下命令就可以在cds/e906/smartl目录下生成CDS工程所需要的文件
yoc cds e906 smartl
#对于soc_xxx示例，在对应demo目录下输入以下命令就可以在cds/e906/smartl/rtthread目录下生成CDS工程所需要的文件
yoc cds e906 smartl rtthread

#在当前demo目录下生成所有cpu平台的CDS工程
yoc cds -f ../../components/xuantie_cpu_sdk/xt_rtos_sdk.csv

#在solutions目录下生成指定demo的所有cpu平台的CDS工程
yoc cds -f ../components/xuantie_cpu_sdk/xt_rtos_sdk.csv -s <demo>

#在solutions目录下生成全部demo所有cpu平台的CDS工程
yoc cds -f ../components/xuantie_cpu_sdk/xt_rtos_sdk.csv

#在solutions目录下生成全部demo的某个cpu平台的CDS工程
yoc cds -f ../components/xuantie_cpu_sdk/xt_rtos_sdk.csv -c <cpu>

```

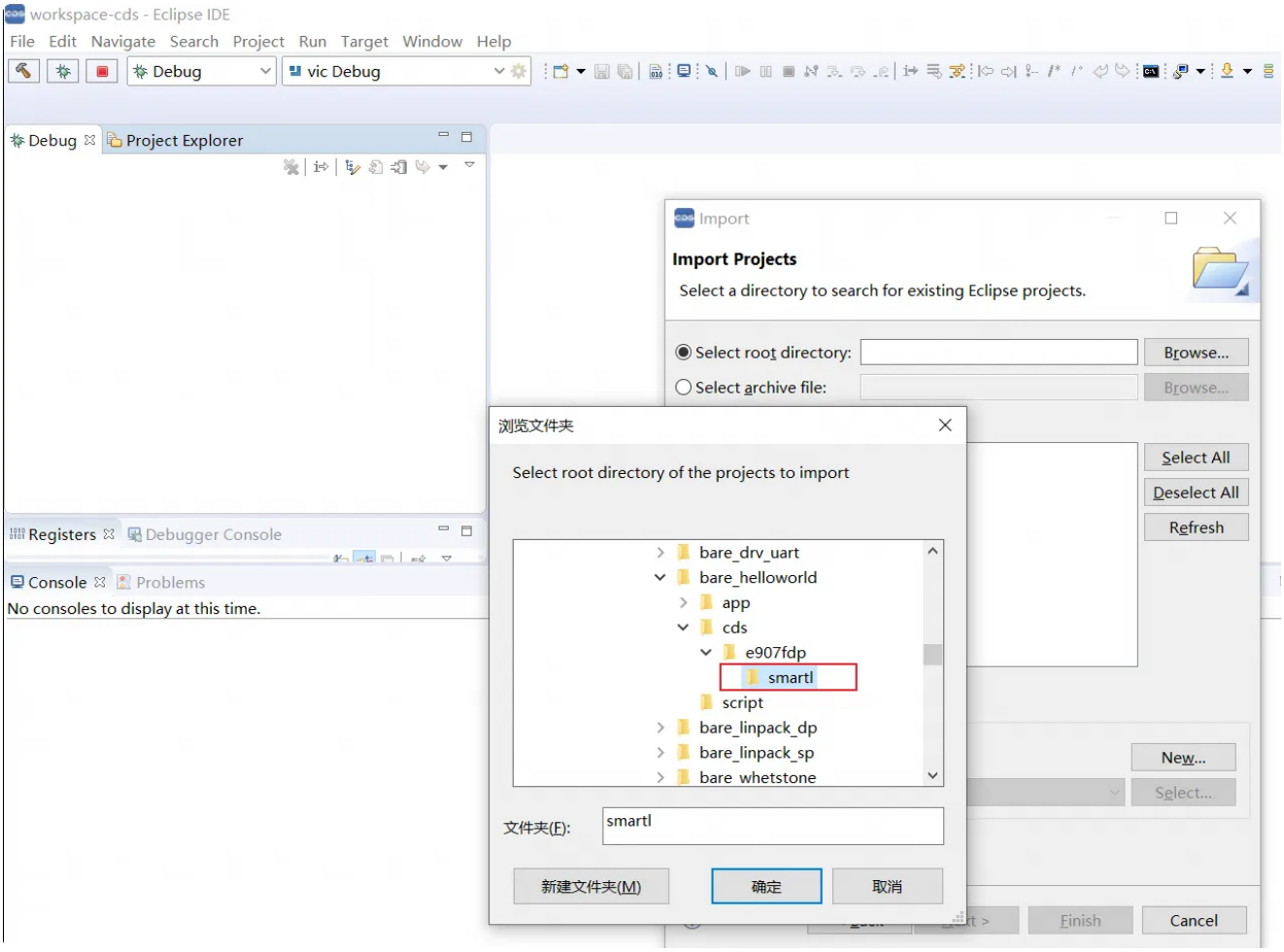
以RTOS-SDK-E907FDP-V2.x.x.zip包中的bare_helloworld示例工程基于CDS编译运行步骤参考如下：

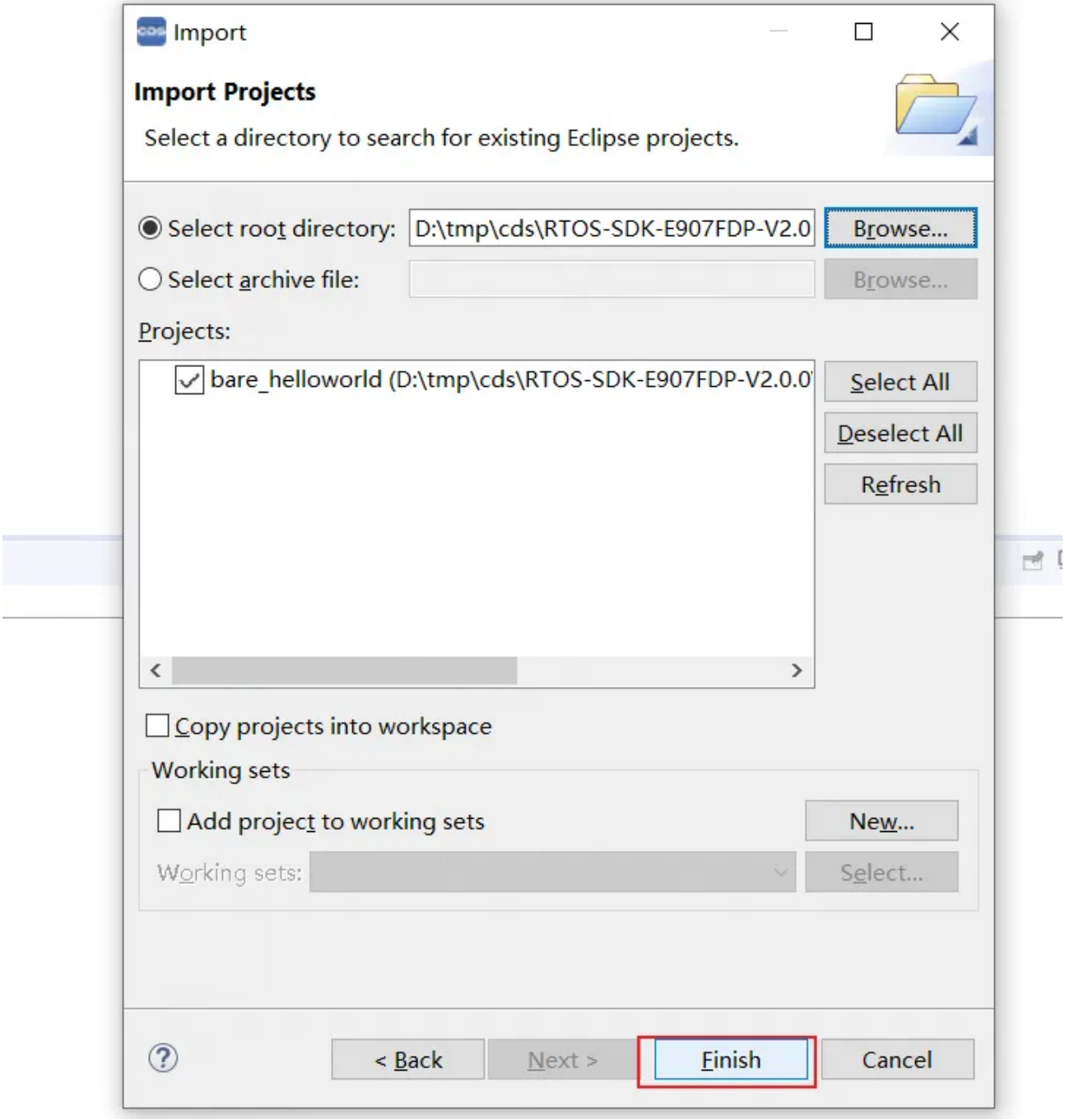
2.4.2.1. 工程导入

通过File->Import导入工程，参考如下：



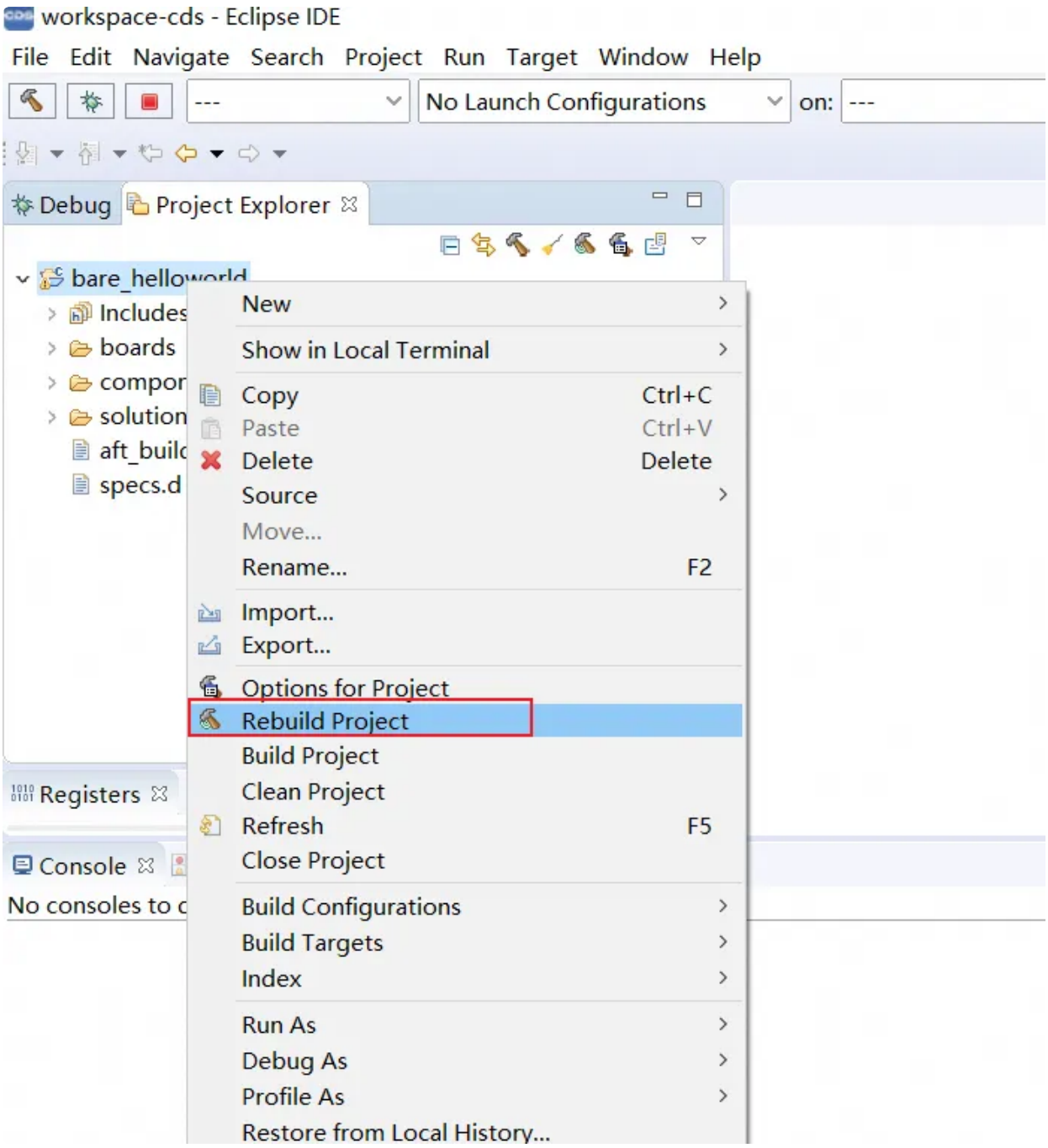
选择helloworld文件夹下的CDS，如下：


workspace-cds - Eclipse IDE
File Edit Navigate Search Project Run Target Window Help
Debug
vic Debug
Debug
Project Explorer
Registers
Debugger Console
Console
Problems
No consoles to display at this time.
Import
Import Projects
Select a directory to search for existing Eclipse projects.
Select root directory:
Browse...
Select archive file:
Browse...
Select All
Deselect All
Refresh
New...
Select...
Finish
Cancel
浏览文件夹
Select root directory of the projects to import
bare_drv_uart
bare_helloworld
app
cds
e907fdp
smartl
script
bare_linpack_dp
bare_linpack_sp
bare_whetstone
文件夹(F):
smartl
新建文件夹(M)
确定
取消

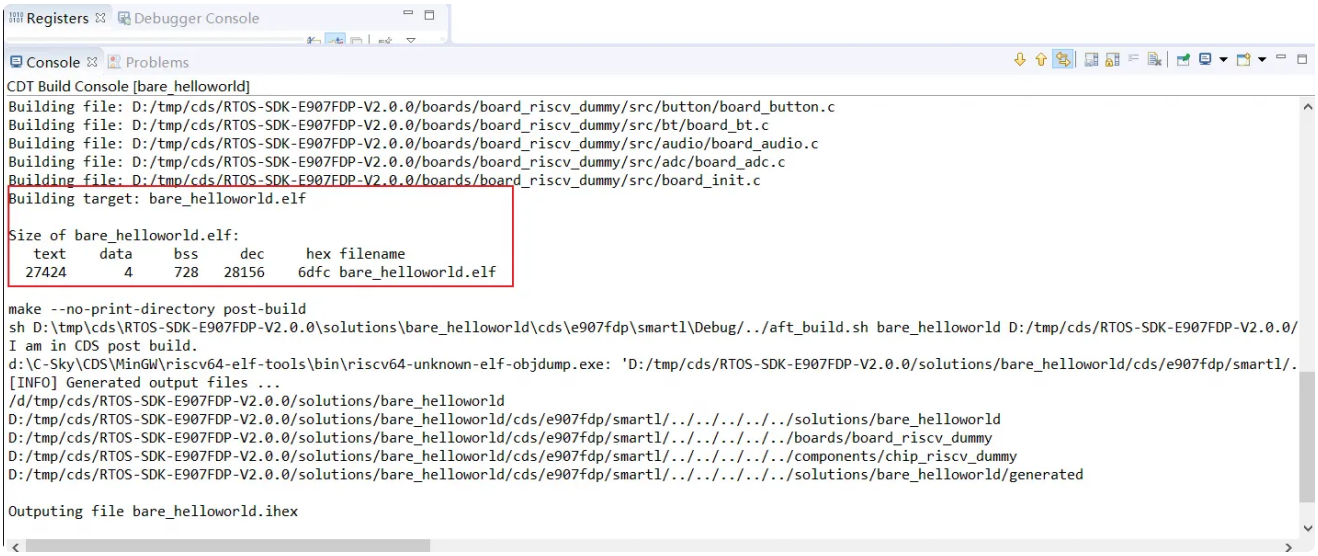


2.4.2.2. 配置&编译

右键点击工程，选择rebuild编译，如下：

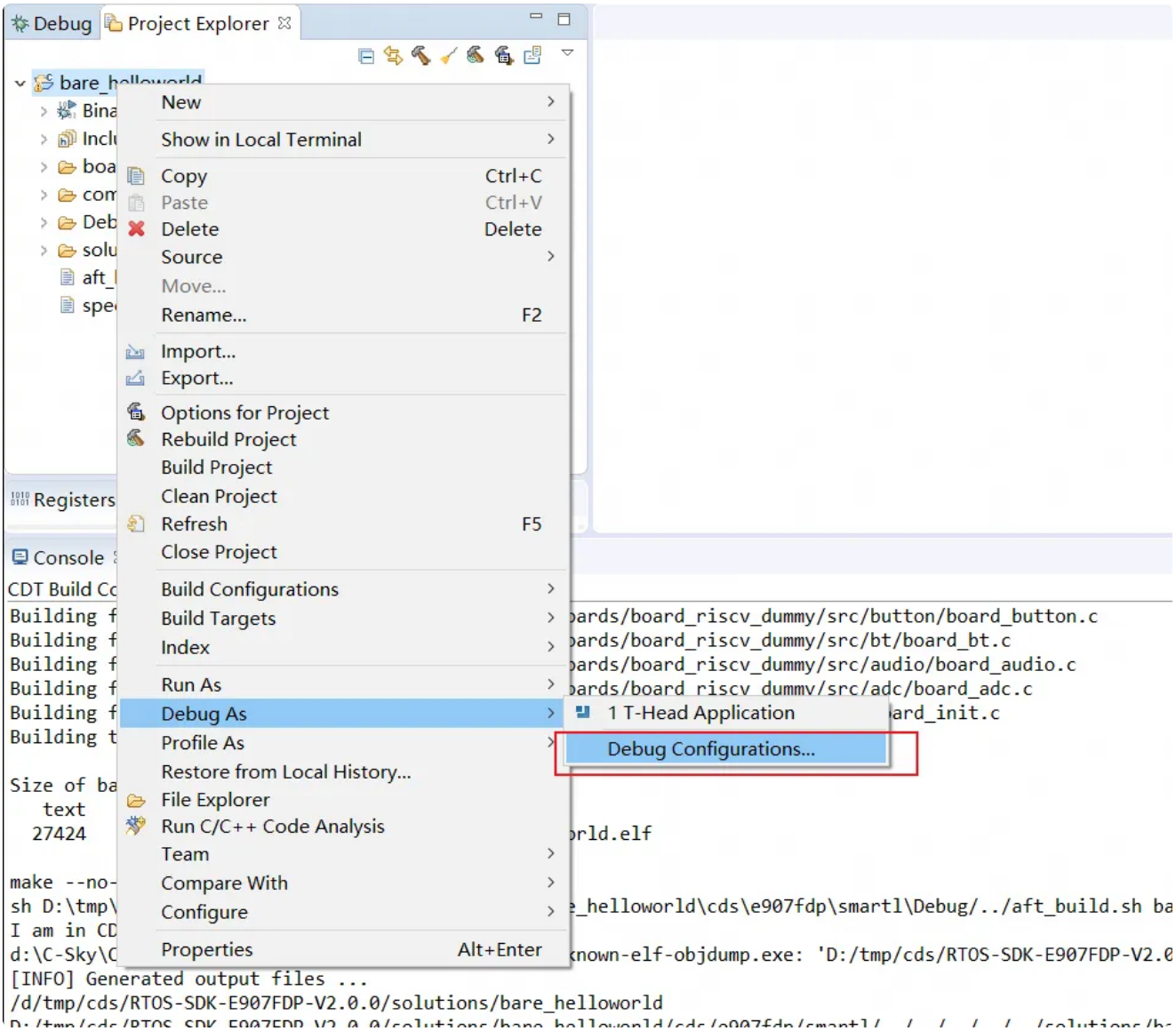


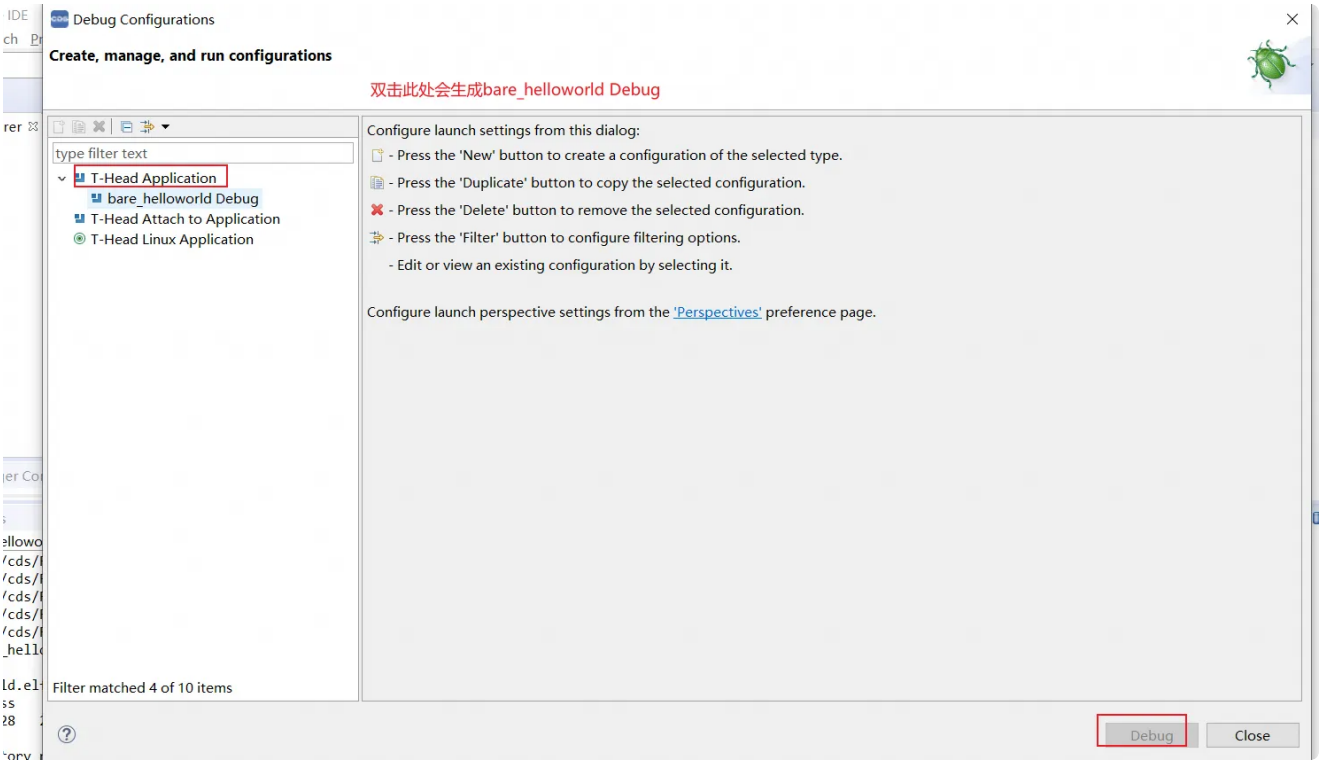
编译成功后，结果参考如下：

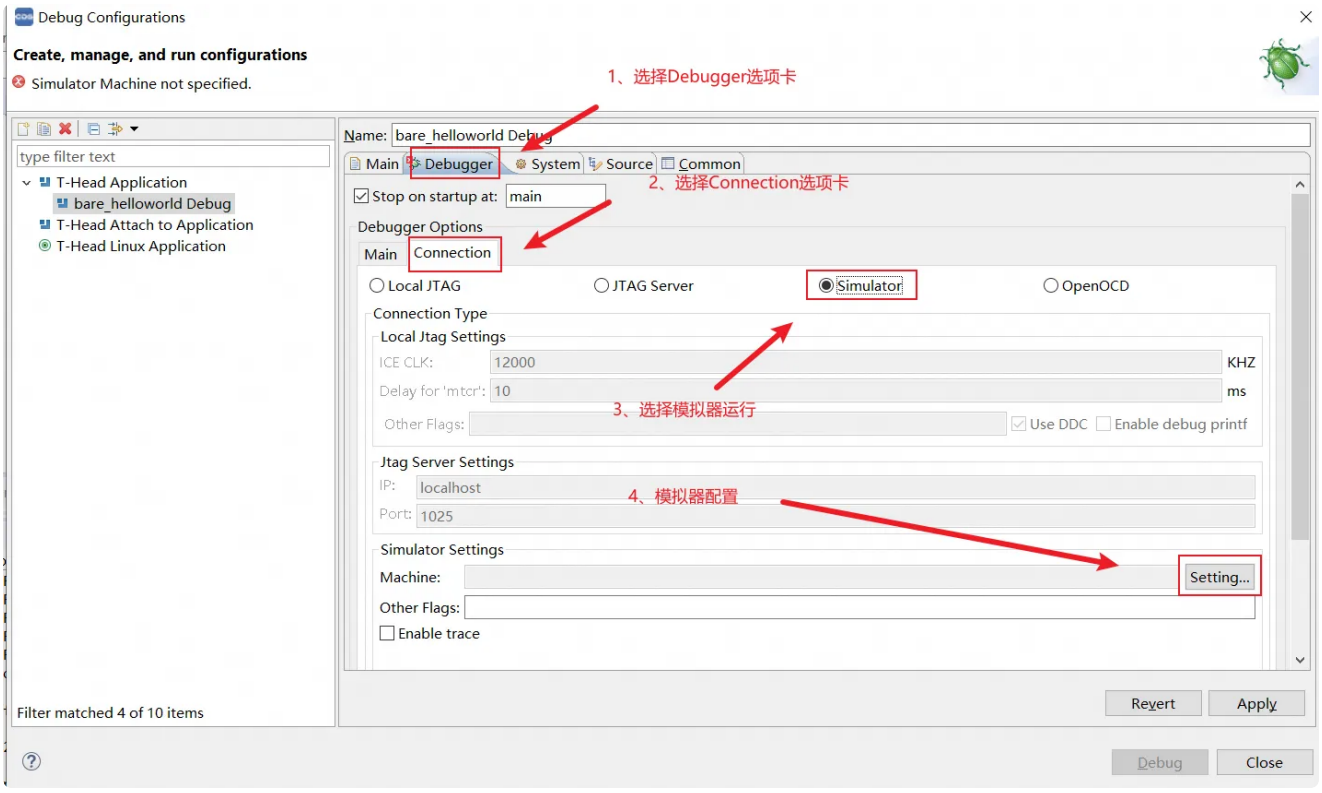


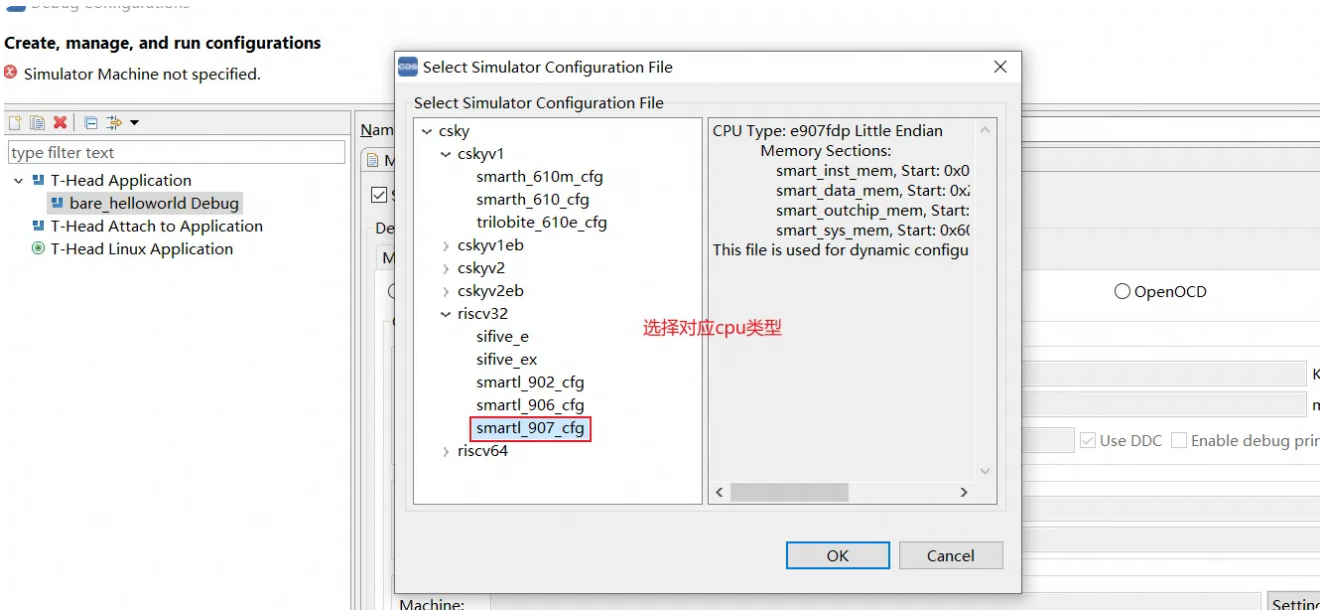
2.4.2.3. 模拟器配置&运行

右键工程，选择Debug Config...，如下：

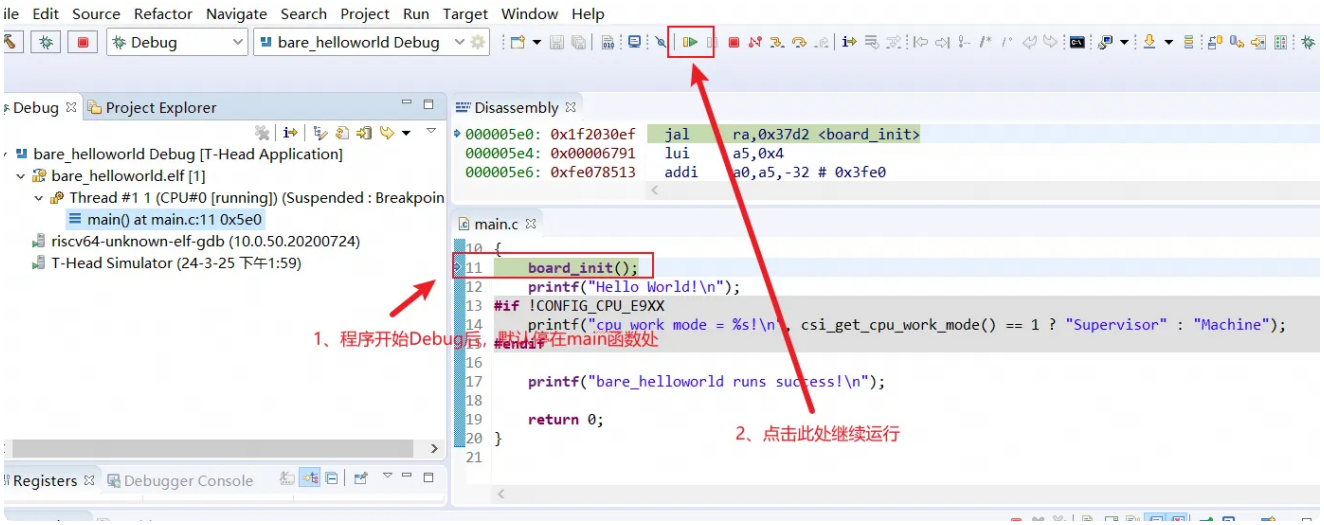

Debug
Project Explorer
bare_helloworld
New
Show in Local Terminal
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Source
Move...
Rename... F2
Import...
Export...
Options for Project
Rebuild Project
Build Project
Clean Project
Refresh F5
Close Project
Build Configurations
Build Targets
Index
Run As
Debug As
Profile As
Restore from Local History...
File Explorer
Run C/C++ Code Analysis
Team
Compare With
Configure
Properties Alt+Enter
1 T-Head Application
Debug Configurations...
Registers
Console
CDT Build Co
Building f
Building f
Building f
Building f
Building f
Building t
Size of ba
text
27424
make --no-
sh D:\tmp\
I am in CD
d:\C-Sky\C
[INFO] Generated output files ...
/d/tmp/cds/RTOS-SDK-E907FDP-V2.0.0/solutions/bare_helloworld
boards/board_riscv_dummy/src/button/board_button.c
boards/board_riscv_dummy/src/bt/board_bt.c
boards/board_riscv_dummy/src/audio/board_audio.c
boards/board_riscv_dummy/src/adc/board_adc.c
ard_init.c
orld.elf
e_helloworld\cds\e907fdp\smartl\Debug/../aft_build.sh ba
known-elf-objdump.exe: 'D:/tmp/cds/RTOS-SDK-E907FDP-V2.0

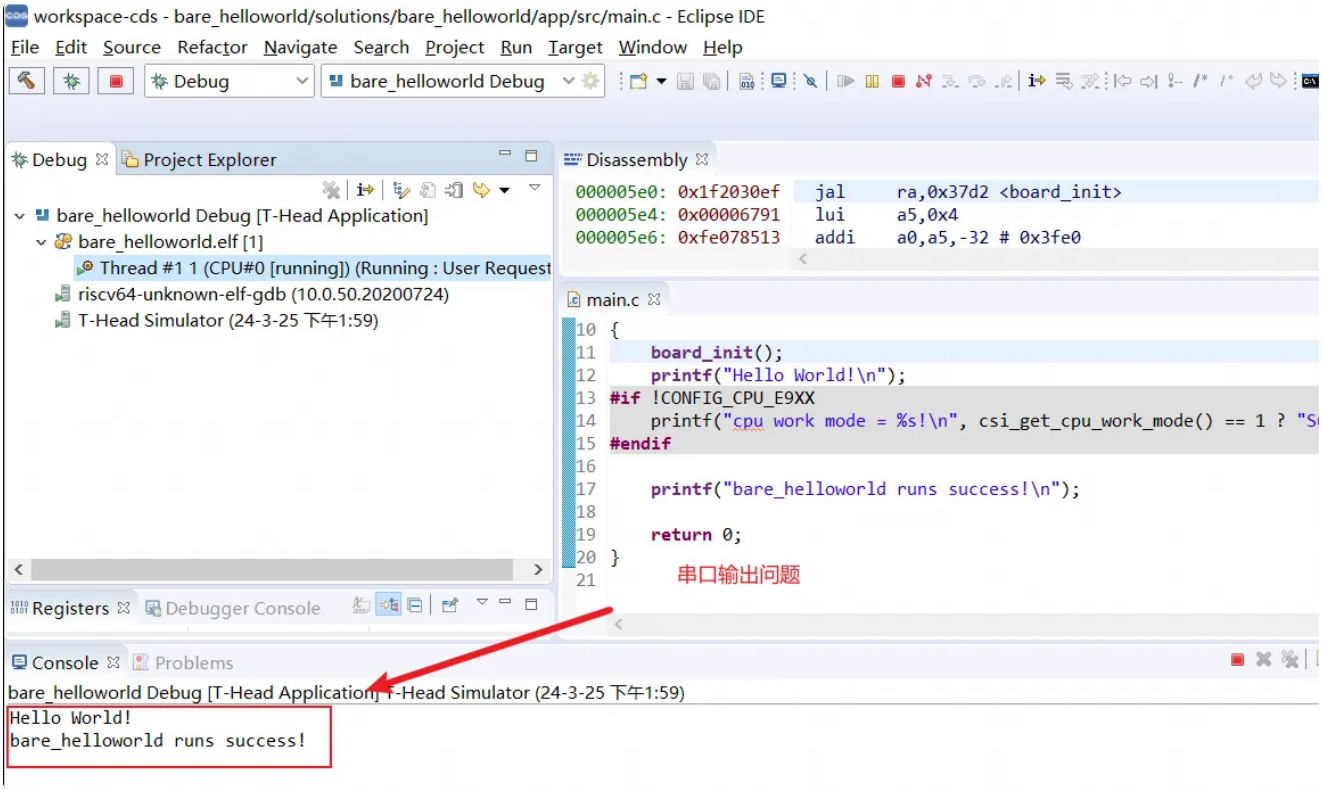

Debug Configurations
Create, manage, and run configurations
双击此处会生成bare_helloworld Debug
type filter text
T-Head Application
bare_helloworld Debug
T-Head Attach to Application
T-Head Linux Application
Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.
Configure launch perspective settings from the 'Perspectives' preference page.
Filter matched 4 of 10 items
Debug
Close


Debug Configurations
Create, manage, and run configurations
Simulator Machine not specified.
1、选择Debugger选项卡
2、选择Connection选项卡
3、选择模拟器运行
4、模拟器配置
type filter text
T-Head Application
bare_helloworld Debug
T-Head Attach to Application
T-Head Linux Application
Name: bare_helloworld Debug
Main
Debugger
System
Source
Common
Stop on startup at: main
Debugger Options
Main
Connection
Local JTAG
JTAG Server
Simulator
OpenOCD
Connection Type
Local Jtag Settings
ICE CLK: 12000 KHZ
Delay for 'mtcr': 10 ms
Other Flags:
Use DDC
Enable debug printf
Jtag Server Settings
IP: localhost
Port: 1025
Simulator Settings
Machine:
Setting...
Other Flags:
Enable trace
Revert
Apply
Filter matched 4 of 10 items
Debug
Close



确认模拟器cpu配置(smartl_907_cfg适用于玄铁e907/e907f/e907fd/e907p/e907fp/e907fdp，其他类似)后，开始Debug，如下：





2.4.3. CDK编译&运行

通过玄铁官方网站下载的ZIP形态SDK包默认在某示例文件夹下生成了对应玄铁CPU型号及默认RTOS类型的CDK工程，用户可参考下述小节内容编译运行。通过yoc命令下载的SDK示例中需要使用bat脚本根据相关参数重建CDK工程(更新project.cdkproj工程文件)，用户可参考以下命令在Windows命令行下生成：

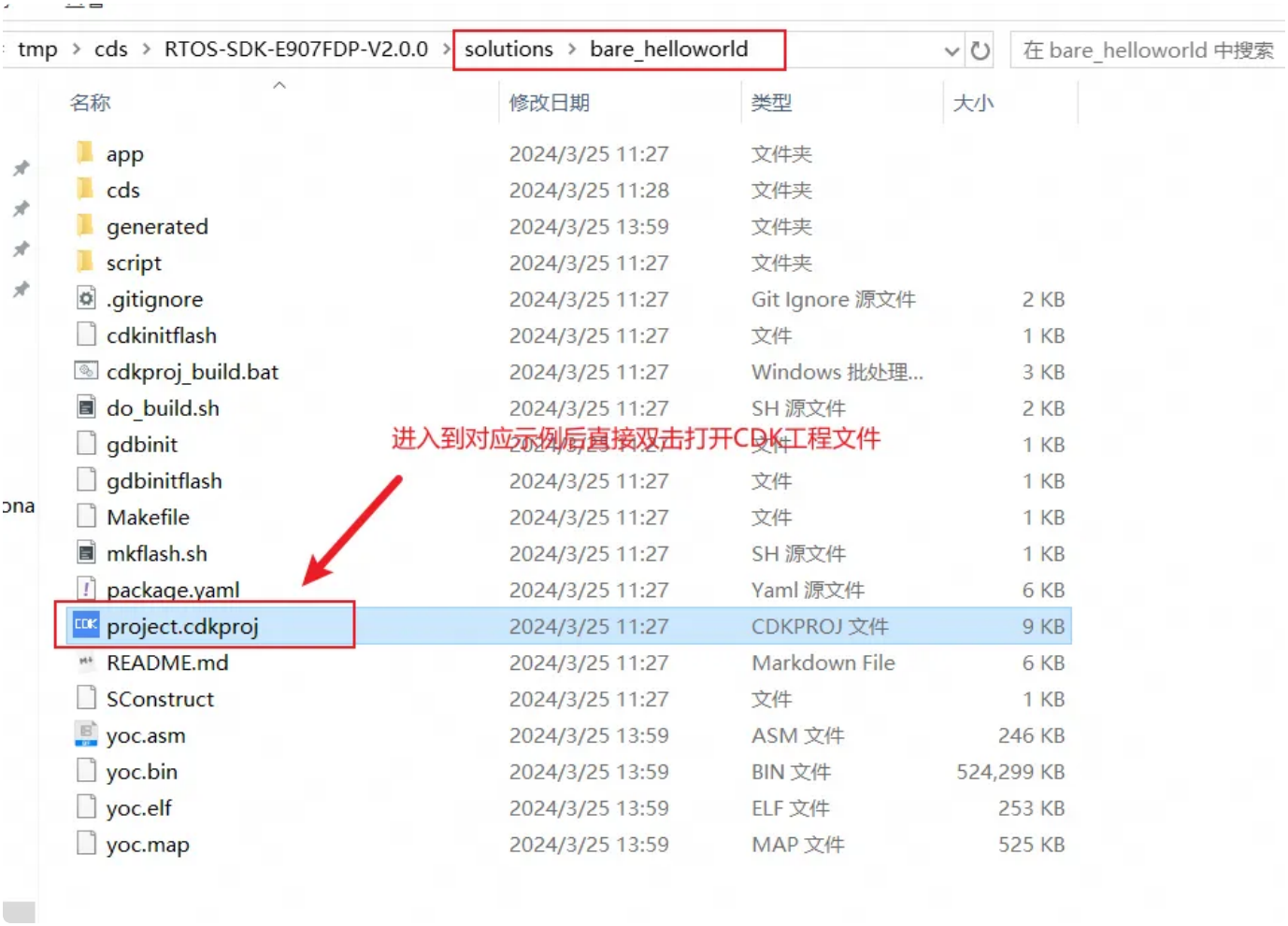
```Bash
#CDK工程需要通过cdkproj_build.bat脚本重建
##对于bare_xxx或mcu_xxx示例，在对应demo目录下参考命令
cdkproj_build.bat e906 smartl
##对于soc_xxx示例，在对应demo目录下参考命令
cdkproj_build.bat e906 smartl rtthread
```

由于玄铁CDK IDE工具主要面向mcu领域开发，所以当前仅支持玄铁Exx系列处理器。用户可基于此在Windows平台下开发，编译出来的elf可以基于CDK内置的玄铁QEMU工具或FPGA平台运行(部分示例虽然基于CDK可以编译，但无法基于QEMU运行，具体请参考第3节说明)。

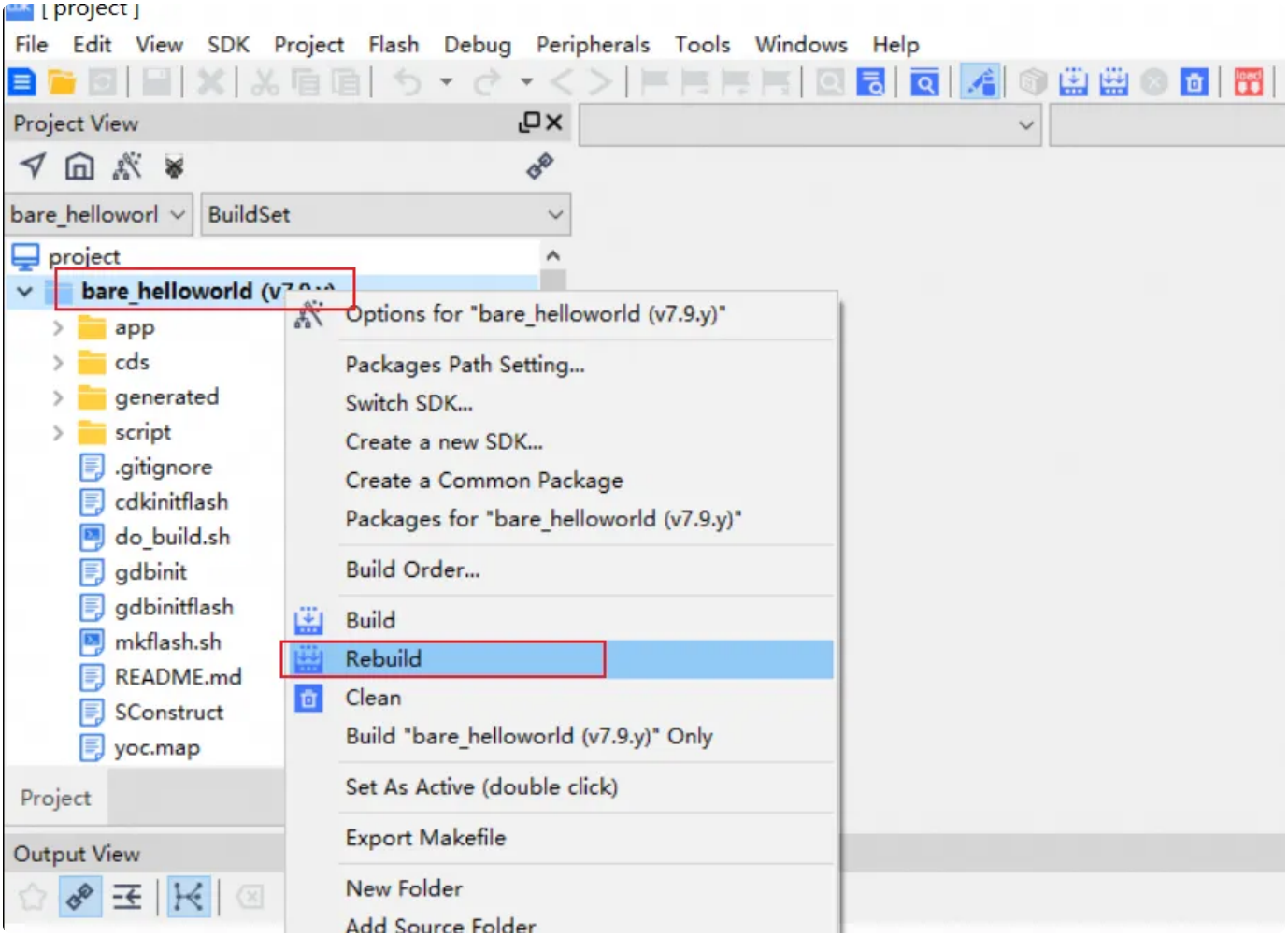
以RTOS-SDK-E907FDP-V2.x.x.zip包中的bare_helloworld示例工程基于CDK编译运行步骤参考如下：

2.4.3.1. 工程打开&编译

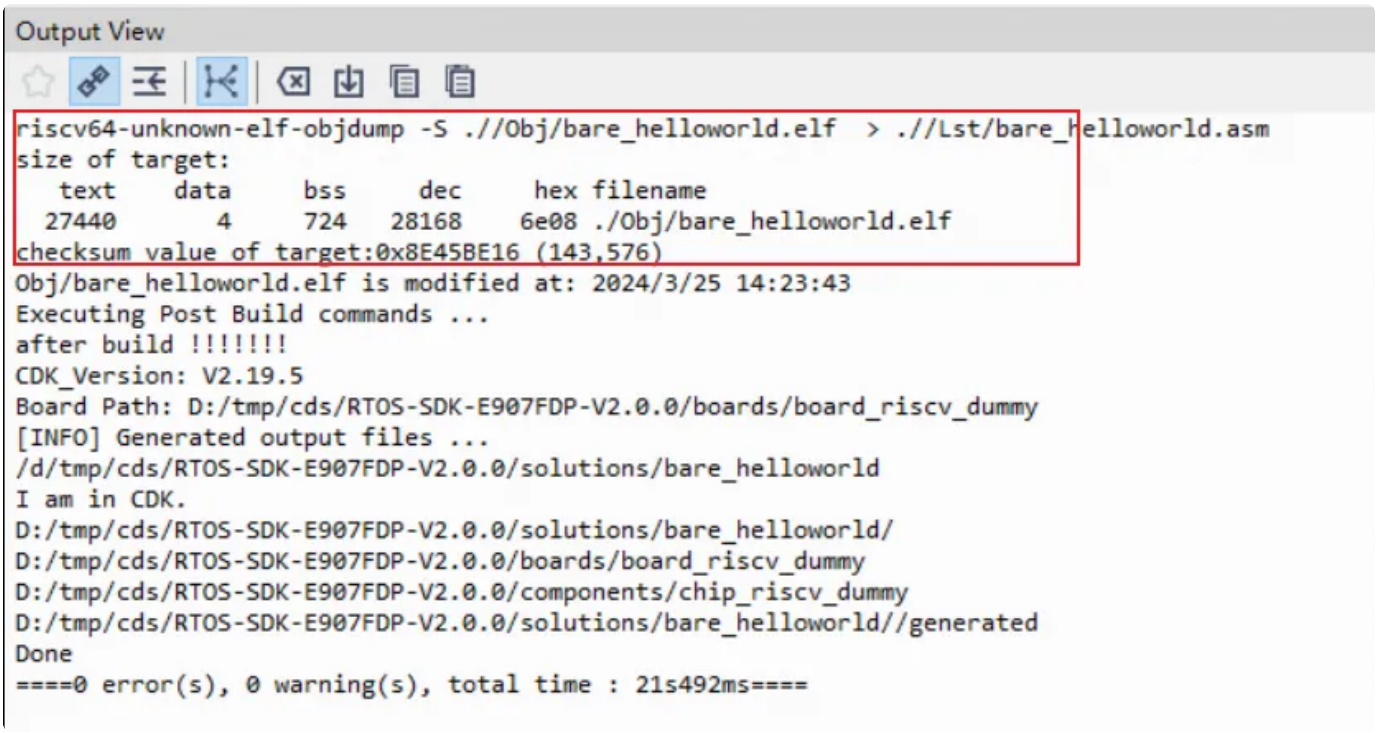
CDK工具的下载请参考第6节相关说明。安装好后，双击对应示例的工程文件打开，如下：



右键相应的工程，选择Rebuild编译构建：



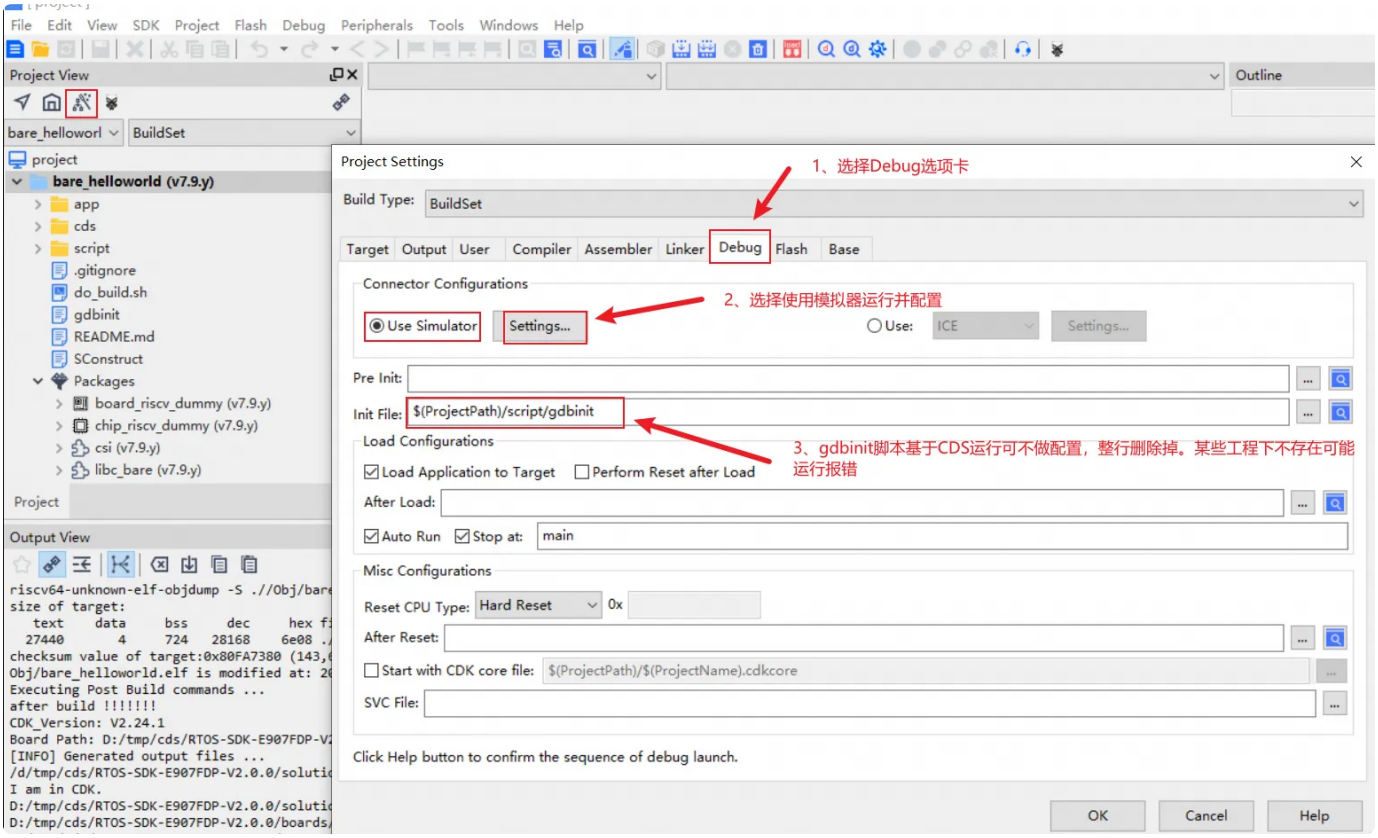
当出现如下编译输出时，表明已编译成功：



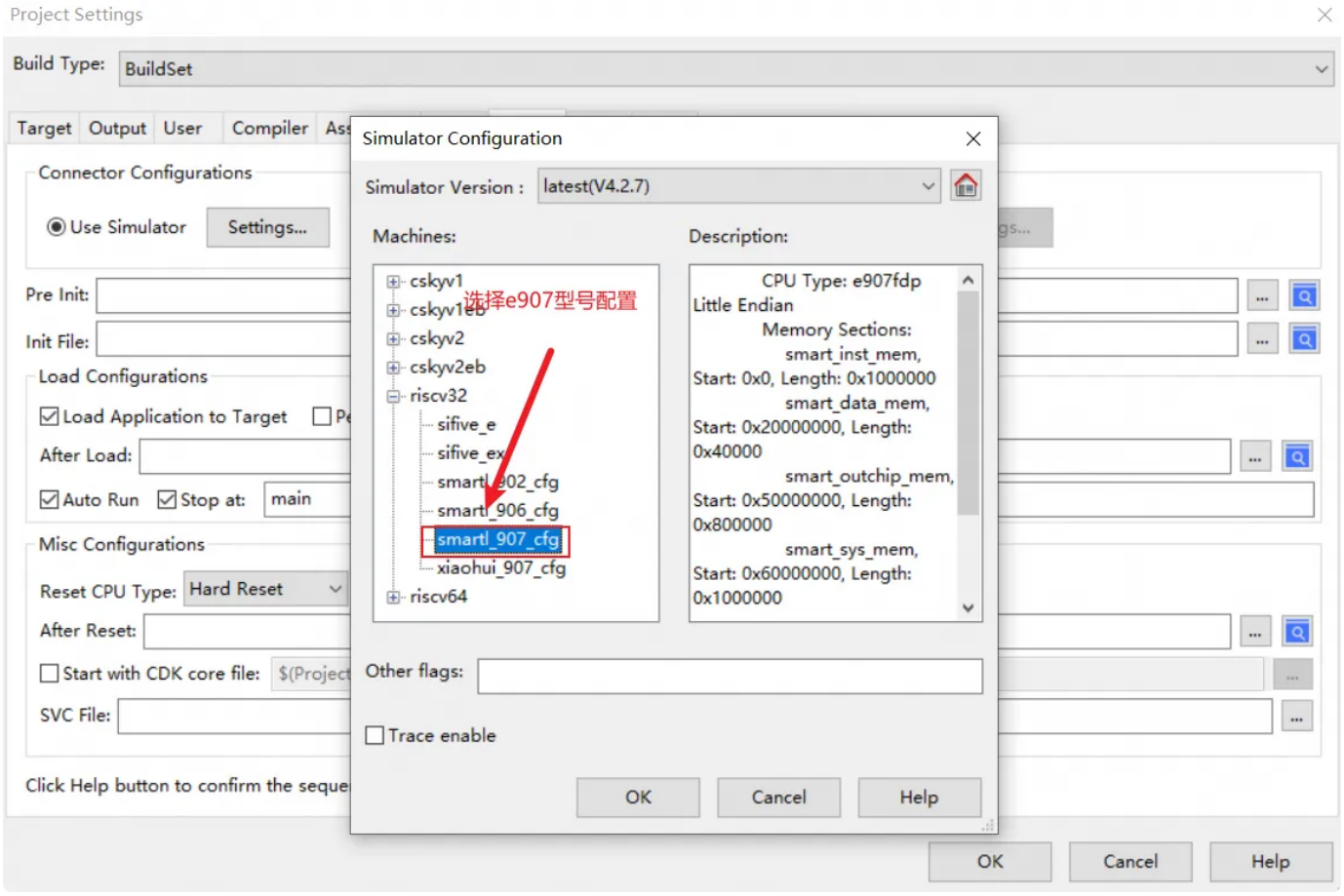
2.4.3.2. 模拟器配置&运行

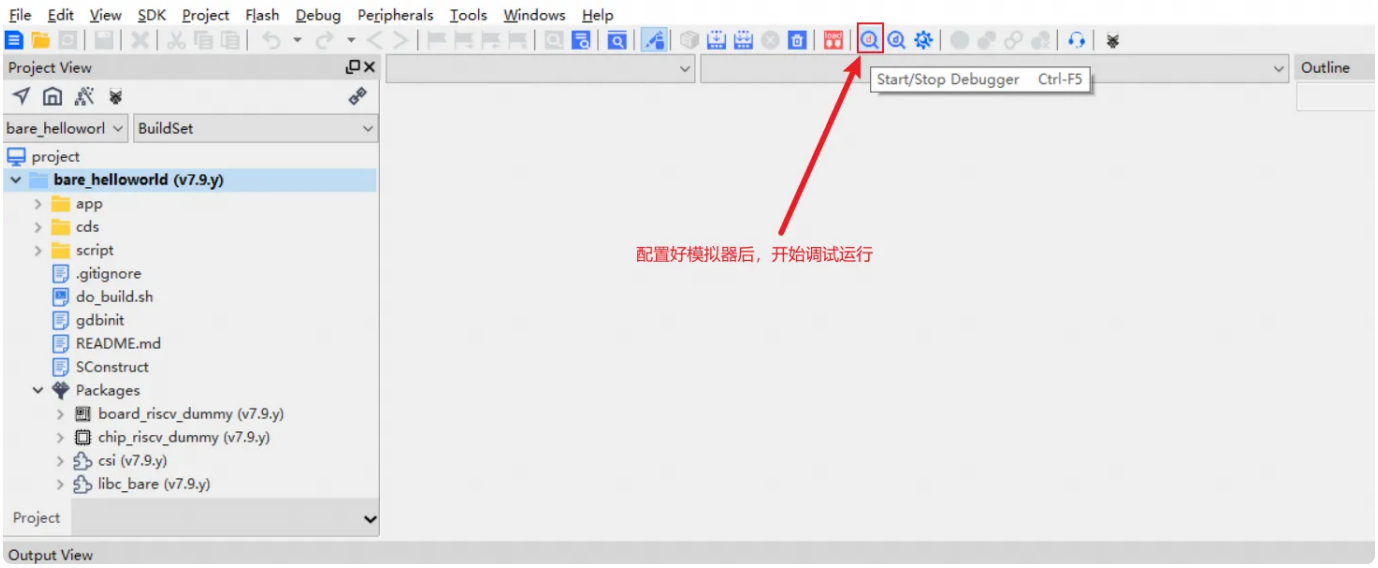
编译出来的elf可直接基于CDK工具中内置了玄铁QEMU工具或参考2.4.1节中基于FPGA平台运行。基于内置的QEMU工具配置&运行参考如下：

选择下图左上的红框进入到工程设置，参考如下：

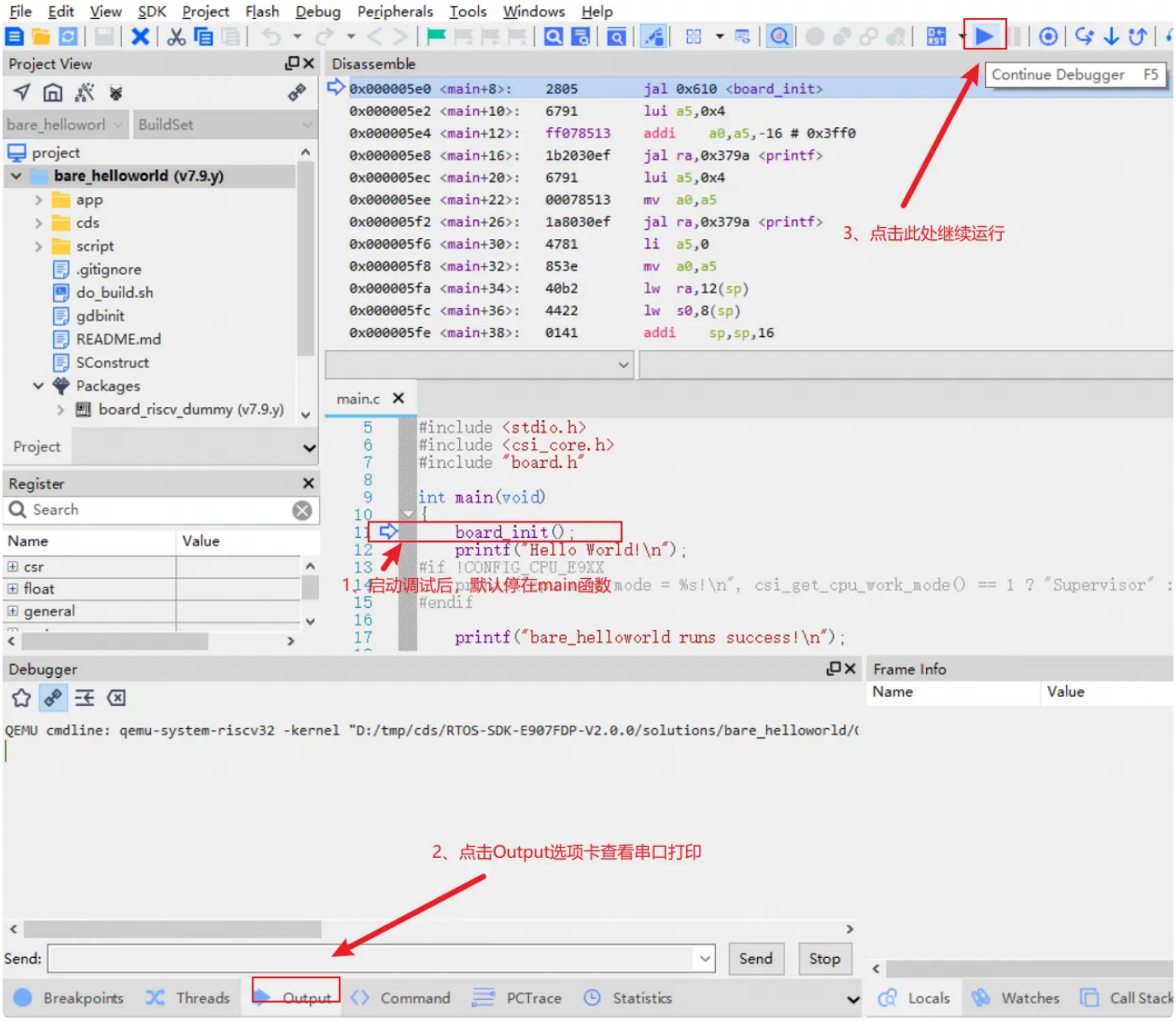


选择E907(smartl平台)配置，如下：

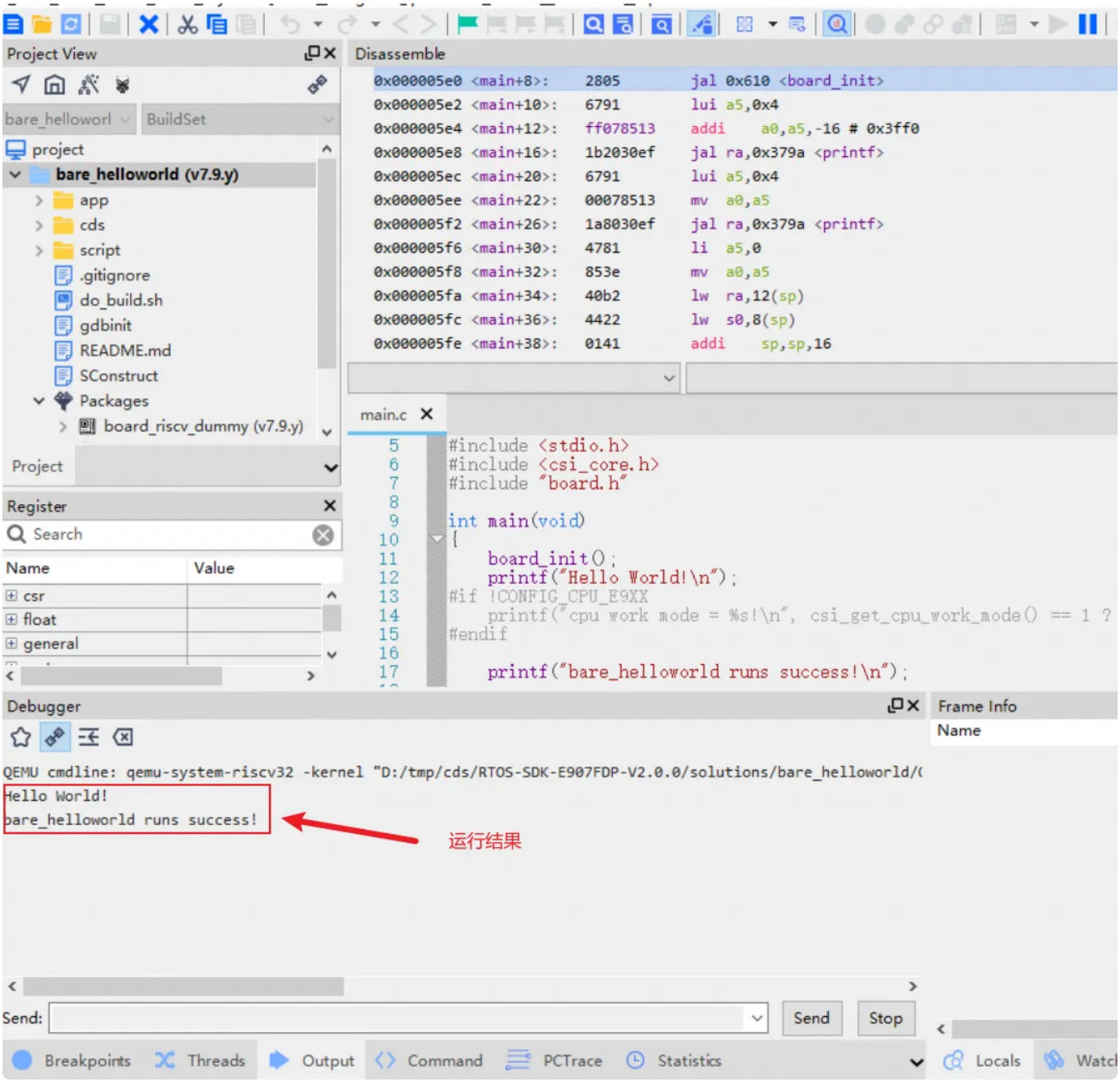




配置好模拟器环境后，开始调试运行，如下：



模拟器运行结果参考如下：



注意：solutions下的soc_xxx_xxx相关示例采用默认RTOS类型。如果需要切换内核类型，可参考README.md文件通过bat脚本重新构建CDK工程。

3. 示例说明

玄铁RTOS SDK中提供了driver、benchmark、kernel相关使用示例参考。由于FPGA平台和QEMU在某些功能上有差异等原因(如ecc、tcm等功能难以模拟)，某些示例并不同时支持FPGA平台和QEMU上的编译或运行。相关功能支持情况请参考3.1节中的说明。

3.1. 功能支持情况

玄铁RTOS SDK中的示例大致遵循以下规则：

- 所有示例在Linux命令行/CDS下均可编译，但并不基于QEMU均可运行
- CDK仅支持玄铁E9xx系列编译
- 部分示例由于跟硬件强相关，QEMU中未模拟，所以仅支持部分示例的运行
- CDK/CDS均内置了玄铁QEMU模拟器，编译出来的elf均可在对应的FPGA平台上通过JTAG工具下载运行

所有示例功能的具体支持情况请参考如下表格，其中第二列表明当前示例所支持的cpu型号，第三列表明该示例是否可以基于玄铁QEMU运行，第四列表明该示例是否可以基于CDK编译：

示例工程	支持处理器型号	QEMU	CDK
bare_helloworld	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
bare_cpp_demo	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y

bare_drv_uart	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
bare_drv_timer	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	N	Y
bare_core_vic	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
bare_core_ecc	c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	N	N
bare_core_tcm	e907/e907f/e907fd/e907p/e907fp/e907fdp	N	Y
bare_core_lockup	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp	N	Y

bare_core_nmi	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp	N	Y
bare_core_wfe	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp	N	Y
bare_core_wfi	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp	N	Y
bare_core_dsp	e906p/e906fp/e906fdp/e907p/e907fp/e907fdp	Y	Y
bare_core_vector	c906fdv/c907fdv/c907fdvm/c907fdv–rv32/c907fdvm–rv32/c908v/c920/c920v2/r920	Y	N
bare_core_matrix	c907fdvm	Y	N
bare_coremark	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	N	Y
bare_dhrystone	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	N	Y

bare_linpack_dp	e906fd/e906fdp/e907fd/e907fdp/c906fd/c906fdv/c907fd/c907fdv/c907fdvm/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c910/c910v2/c920/c920v2/r910/r920	N	Y
bare_linpack_sp	e906f/e906fd/e906fp/e906fdp/e907f/e907fd/e907fp/e907fdp/c906fd/c906fdv/c907fd/c907fdv/c907fdvm/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c910/c910v2/c920/c920v2/r910/r920	N	Y
bare_whetstone	e906f/e906fd/e906fp/e906fdp/e907f/e907fd/e907fp/e907fdp/c906fd/c906fdv/c907fd/c907fdv/c907fdvm/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c910/c910v2/c920/c920v2/r910/r920	N	Y
mcu_rtthread_helloworld	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_rtthread_mutex	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y

mcu_rtthread_task	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_rtthread_message_q	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_rtthread_event	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_rtthread_timer	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y

mcu_rtthread_time	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_rtthread_sem	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_rtthread_cpp	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_rtthread_smp	c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	N
mcu_freertos_helloworld	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y

mcu_freertos_mutex	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_freertos_task	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_freertos_message_q	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_freertos_event	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y

mcu_freertos_timer	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_freertos_time	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_freertos_sem	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
mcu_freertos_cpp	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y

soc_helloworld	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
soc_kernel_mutex	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
soc_kernel_task	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
soc_kernel_message_q	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907–rv32/c907fd–rv32/c907fdv–rv32/c907fdvm–rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y

soc_kernel_event	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
soc_kernel_timer	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
soc_kernel_time	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
soc_kernel_sem	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y

soc_cpp_demo	e902/e902m/e902t/e902mt/e906/e906f/e906fd/e906p/e906fp/e906fdp/e907/e907f/e907fd/e907p/e907fp/e907fdp/c906/c906fd/c906fdv/c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	Y
soc_smp_demo	c907/c907fd/c907fdv/c907fdvm/c907-rv32/c907fd-rv32/c907fdv-rv32/c907fdvm-rv32/c908/c908v/c908i/c910/c910v2/c920/c920v2/r910/r920	Y	N
soc_core_dsp	e906p/e906fp/e906fdp/e907p/e907fp/e907fdp	Y	Y
soc_core_vector	c906fdv/c907fdv/c907fdvm/c907fdv-rv32/c907fdvm-rv32/c908v/c920/c920v2/r920	Y	N
soc_core_matrix	c907fdvm	Y	N

3.2. 面向裸系统级别示例

面向裸系统级别的示例提供了驱动、中断控制器、低功耗、tcm等功能的使用参考。用户可基于相关代码移植到自有软件框架中。

3.2.1. bare_helloworld

该示例位于solutions/bare_helloworld，其是一个裸系统最小工程。

其中对于玄铁E9xx系列仅支持M态，C9xx&R9xx仅支持M&S态(默认M态)，可通过修改package.yaml文件，使能S态支持(开启CONFIG_RISCV_SMODE配置)，如下：

```
def_config:
  CONFIG_KERNEL_NONE: 1
  CONFIG_DEBUG: 1
  CONFIG_RISCV_SMODE: 1
```

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果参考如下：

```
Hello World!
cpu work mode = Machine!
bare_helloworld runs success!
```

其中第二行打印表明当前运行在M态。

注意：当前玄铁QEMU暂不支持S态运行

3.2.2. 裸驱相关示例

裸驱相关示例提供了常见的外设使用示例，包括串口和定时器的使用参考。相关示例当前是基于CSI2.0接口实现。

3.2.2.1. bare_drv_uart

该示例位于solutions/bare_drv_uart，主要提供串口查询和中断两种模式的打印输出功能。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果参考如下：

```
bare_drv_uart demo start
==uart_test_sync_mode start==
ABCDEFGHIJKLMN
ABCDEFGHIJKLMN
==uart_test_sync_mode end==
==uart_test_async_mode start==
ABCDEFGHIJKLMN
==uart_test_async_mode end==
bare_drv_uart success
```

3.2.2.2. bare_drv_timer

该示例位于solutions/bare_drv_timer，用于演示定时器驱动的使用。该示例核心功能为开始时设置了一个1s的定时器，然后等待2s检查定时器中断是否触发，正常情况下3s后打印成功运行结果。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果参考如下：

```Bash
bare_drv_timer demo start!
bare_drv_timer success
```

3.2.3. cpu core相关示例

cpu core相关示例提供了玄铁处理器中关于中断、中断嵌套、低功耗、NMI、TCM、ECC、DSP等功能的使用参考。

3.2.3.1. bare_core_vic

该示例位于solutions/bare_core_vic，用于演示中断控制器、中断嵌套等功能的使用。示例中开启了两个不同优先级的定时器，对于支持中断嵌套的玄铁处理器，高优先级的定时器中断会抢占低优先中断。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
set timer 0 's interrrupt priority with 2 (lower priority).
initialize timer 1, set it's interrupt priority : 3 (higher priority).
start reload timer 0 with lower interrupt priority, and it will enter an en
dless loop callback
start reload mode timer 1 with higher interrupt priority.
I am timer 0's callback in an endless loop.
I am timer 1's with higher interrupt priority than timer 0.
Example run successfully!
core-vic runs successfully!
```

基于玄铁内部smartl FPGA(以e907fdp为例)平台上的默认运行结果如下：

```Bash
bare_core_vic demo start!
set timer 0 's interrrupt priority with 2 (lower priority).
initialize timer 1, set it's interrupt priority : 3 (higher priority).
start reload timer 0 with lower interrupt priority, and it will enter an endless loop callback
start reload mode timer 1 with higher interrupt priority.
I am timer 1's with higher interrupt priority than timer 0.
I am timer 0's callback in an endless loop.
bare_core_vic runs success!
```

注意：中断嵌套功能当前仅支持玄铁E9xx处理器。

3.2.3.2. bare_core_ecc

该示例位于solutions/bare_core_ecc，用于演示L1&L2 ECC校验机制的使用。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
bare_core_ecc demo start!
before L1 cache ecc inject, mcer = 0x0
after L1 cache ecc inject, mcer = 0x8000000081200000
l1 ecc runs successfully!
before L2 cache ecc inject, mcer2 = 0x0
after L2 cache ecc inject, mcer2 = 0x813000000e000000
l2 ecc runs successfully!
```

注意：

- 当前仅玄铁C9xx & R9xx部分型号支持ecc功能。
- 当运行出现失败打印时，可能的原因是在制作bit时未使能L1 or L2 ecc功能，属于正常现象。此时可通过jtag连接时的打印信息确认

3.2.3.3. bare_core_dsp

该示例位于solutions/bare_core_dsp，用于演示玄铁处理器P扩展使用方式。示例程序中会调用libcsi_xt900p32fd_dsp.a(采用P扩展指令封装，实际存储在components/csi/csi2/dsp文件夹中)库中的csi_abs_f32函数计算。

基于玄铁内部smartl FPGA(以e907fdp为例)平台上的默认运行结果如下：

```
bare_core_dsp demo start!
bare_core_dsp runs success!
```

注意：该示例仅支持玄铁处理器带P扩展的型号上编译运行

3.2.3.4. bare_core_lockup

该示例位于solutions/bare_core_lockup，用于演示玄铁exx cpu锁定功能。示例程序中首先使用TIMER4作为NMI中断的触发源(3s后触发)，然后访问非法地址进入到异常流程，之后在异常处理流程中再次访问非法地址使得cpu进入锁定状态，最后等待NMI中断触发。

基于玄铁内部smartl FPGA(以e907fdp为例)平台上的默认运行结果如下：

```
bare_core_lockup demo start!
first: access illegal address 0xf1234568, trigger exception!
CPU Exception: NO.0x5
x1: 00003AB2    x2: 20002380    x3: 20000004    x4: 00000000
x5: 000042B0    x6: 0000000F    x7: 200000C4    x8: 200023A0
x9: 00000000    x10: 0000003D   x11: 20002344   x12: 0000003D
x13: FFFFFFFF   x14: 000006B2   x15: FFFFFFFF   x16: 00030D40
x17: 00000000   x18: 00000000   x19: 00000000   x20: 00000000
x21: 00000000   x22: 00000000   x23: 00000000   x24: 00000000
x25: 00000000   x26: 00000000   x27: 00000000   x28: 00000000
x29: 00000000   x30: 00000000   x31: 00000000
mepc   : 00003AC8
mstatus: 00003800
entry exception callback: my_trap_c
second: access illegal address 0xf1234568 again, trigger lockup! and wait for nmi!
nmi has happened
bare_core_lockup runs success!
```

注意：该示例当前仅支持在玄铁Exx处理器上编译运行

3.2.3.5. bare_core_nmi

该示例位于solutions/bare_core_nmi，用于演示玄铁处理器非屏蔽中断的使用方式。示例程序中运行所依赖的bit将TIMER4作为NMI中断的触发源。代码中首先初始化TIMER4并设为3s后触发中断，然后关闭所有的中断，最后等待3s后NMI中断触发。

基于玄铁内部smartl FPGA(以e907fdp为例)平台上的默认运行结果如下：

```Bash
bare_core_nmi demo start!
nmi has happened
bare_core_nmi runs success!
```

注意：该示例当前仅支持在玄铁Exx处理器上编译运行

3.2.3.6. bare_core_tcm

该示例位于solutions/bare_core_tcm，用于演示玄铁处理器紧耦合存储模块(TCM)的使用方式。示例代码中将tcm_code.c对应的代码段等放到_itcm_code中(参考solutions/bare_core_tcm/gcc_flash_tcm_smartl.ld链接脚本中的用法)。系统启动后将_itcm_code等段中的数据拷贝到ITCM/DTCM指定的地址空间中并使能，最后执行tcm_code.c相关函数。

基于玄铁内部smartl FPGA(以e907fdp为例)平台上的默认运行结果如下：

```Bash
bare_core_tcm demo start!
[9]: execute in tcm region
[8]: execute in tcm region
[7]: execute in tcm region
[6]: execute in tcm region
[5]: execute in tcm region
[4]: execute in tcm region
[3]: execute in tcm region
[2]: execute in tcm region
[1]: execute in tcm region
[0]: execute in tcm region
bare_core_tcm runs success!
```

玄铁处理器对于TCM设计有以下特点：

- ITCM/DTCM的基地址不是固定的，可通过设置对应寄存器灵活配置地址空间。可参考示例代码中csi_dtcm_set_base_addr和csi_itcm_set_base_addr
- ITCM/DTCM的使能需要单独设置对应寄存器。可参考示例代码中csi_dtcm_enable和csi_itcm_enable
- ITCM/DTCM的区域大小有soc设计决定。玄铁提供的参考设计ITCM/DTCM默认均为32KB。具体大小也可通过接口csi_itcm_get_size获取

- solutions/bare_core_tcm/gcc_flash_tcm_smartl.ld链接脚本中ITCM/DTCM的基地址分别设置为0x30000000和0x30008000。这两个地址是虚拟的，不占用实际外设等地址空间。于此同时代码中若使用时，需要与这两个地址对应起来

注意：该示例当前仅支持在玄铁带硬件tcm模块的处理器型号上编译运行

3.2.3.7. bare_core_wfe

该示例位于solutions/bare_core_wfe，用于演示玄铁处理器低功耗(wfi)的使用方式(通过NMI中断唤醒)。示例程序中运行所依赖的bit将TIMER4作为NMI中断的触发源。代码中首先初始化TIMER4并设为3s后触发中断，然后关闭所有的中断后并执行wfi指令进入低功耗，最后等待3s后NMI中断触发唤醒CPU。

基于玄铁内部smartl FPGA(以e907fdp为例)平台上的默认运行结果如下：

```Bash
bare_core_wfe demo start!
execute WFE and enter standby mode:
nmi has happened
bare_core_wfe runs success!
```

注意：该示例当前仅支持在玄铁Exx处理器上编译运行

3.2.3.8. bare_core_wfi

该示例位于solutions/bare_core_wfi，用于演示玄铁处理器低功耗(wfi)的使用方式(通过普通中断唤醒)。示例程序中运行所依赖的bit将TIMER0(普通中断)作为低功耗唤醒的触发源。代码中首先初始化TIMER0并设为3s后触发中断，然后执行wfi指令进入低功耗，最后等待3s后TIMER0中断触发唤醒CPU。

基于玄铁内部smartl FPGA(以e907fdp为例)平台上的默认运行结果如下：

```Bash
bare_core_wfi demo start!
execute wfi and enter standby mode:
wake up by interrupt
bare_core_wfi runs success!
```

注意：该示例当前仅支持在玄铁Exx处理器上编译运行

3.2.3.9. bare_core_vector

该示例位于solutions/bare_core_vector，用于演示玄铁处理器vector扩展使用方式。用户可参考vector汇编相关代码实现其他功能。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
bare_core_vector demo start!
[fp32] gemm start
=====>>native test start.
=====>>native test end.
=====>>vector test start.
===vlen=256
=====>>vector test end.
native time = 116ms, vector time = 30ms. diff = 86ms
[fp32] gemm end, total diff time = 227ms
vector runs successfully!
```

注意：该示例仅支持玄铁处理器带v扩展的型号上编译运行

3.2.3.10. bare_core_matrix

该示例位于solutions/bare_core_matrix，用于演示玄铁处理器matrix扩展使用方式。用户可参考matrix汇编相关代码实现其他功能。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
bare_core_matrix demo start!
[int8] gemm start
[m * k * n = 48 * 48 * 48] best execution time: 0.000ms, inf GFLOPS
[m * k * n = 96 * 96 * 96] best execution time: 3.000ms, 0.59 GFLOPS
[m * k * n = 144 * 144 * 144] best execution time: 5.000ms, 1.19 GFLOPS
[m * k * n = 192 * 192 * 192] best execution time: 11.000ms, 1.29 GFLOPS
[m * k * n = 240 * 240 * 240] best execution time: 20.000ms, 1.38 GFLOPS
[int8] gemm end
matrix runs successfully!
```

注意：该示例仅支持玄铁处理器带matrix扩展的型号上编译运行

3.2.4. benchmark相关示例

benchmark相关的示例提供了常见的处理器性能评估基准，如coremark、whetstone、dhrystone等。用户可借助这些示例评测玄铁处理器实际性能。

注意：benchmark相关示例当前仅支持在fpga上运行，基于QEMU运行无实际参考意义。

3.2.4.1. bare_coremark

该示例位于solutions/bare_coremark，用于评估嵌入式系统处理器的性能，取代过时的dhrystone基准。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
bare_coremark demo start!
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 18429121
Total time (secs): 18.429139
Iterations/Sec   : 162.785680
Iterations       : 3000
Compiler version : GCC10.4.0
Compiler flags   :
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xcc42
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 162.785680 / GCC10.4.0  / STACK
Score (Coremarks/MHz):   3.26
bare_coremark runs success!
```

3.2.4.2. bare_whetstone

该示例位于solutions/bare_whetstone，其用于评估处理器的浮点运算性能。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
bare_whetstone demo start!

Loops: 10000, Iterations: 1, Duration: 30 sec.
C Converted Double Precision Whetstones: 32.8 MIPS
bare_whetstone runs success!
```

注意：该示例当前仅支持在玄铁带硬浮点的处理器上编译运行

3.2.4.3. bare_linpack_sp

该示例位于solutions/bare_linpack_sp，其用于评估计算机系统浮点运算(单精度)性能的基准测试程序。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
bare_linpack_sp demo start!
Unrolled Single Precision Linpack

Unrolled Single Precision Linpack

     norm. resid      resid           machep         x[0]-1        x[n-1]-1
       3.3        1.56606838e-04  1.19209289e-07  2.45571133e-05 -1.22189520e-05
    times are reported for matrices of order   200
      dgefa      dgesl      total       kflops     unit      ratio
 times for array with leading dimension of  201
       0.52       0.01       0.53      10198       0.20       9.48
       0.52       0.01       0.53      10197       0.20       9.48
       0.52       0.01       0.53      10195       0.20       9.48
       0.52       0.01       0.53      10198       0.20       9.48
 times for array with leading dimension of 200
       0.52       0.01       0.53      10214       0.20       9.46
       0.52       0.01       0.53      10221       0.20       9.46
       0.52       0.01       0.53      10210       0.20       9.47
       0.52       0.01       0.53      10212       0.20       9.47
Unrolled Single  Precision 10198 Kflops ; 10 Reps
bare_linpack_sp runs success!

```

注意：该示例当前仅支持在玄铁带单精度硬浮点的处理器上编译运行

3.2.4.4. bare_linpack_dp

该示例位于solutions/bare_linpack_dp，其用于评估计算机系统浮点运算(双精度)性能的基准测试程序。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
bare_linpack_dp demo start!
Unrolled Double Precision Linpack

Unrolled Double Precision Linpack

     norm. resid      resid           machep         x[0]-1        x[n-1]-
1
       1.9        8.46778493e-14  2.22044605e-16 -1.11799459e-13 -9.603429
10e-14
    times are reported for matrices of order   100
      dgefa      dgesl      total       kflops     unit      ratio
 times for array with leading dimension of  101
       0.08       0.00       0.08       8746       0.23       1.40
       0.08       0.00       0.08       8739       0.23       1.40
       0.08       0.00       0.08       8748       0.23       1.40
       0.08       0.00       0.08       8748       0.23       1.40
 times for array with leading dimension of 100
       0.08       0.00       0.08       8683       0.23       1.41
       0.08       0.00       0.08       8689       0.23       1.41
       0.08       0.00       0.08       8699       0.23       1.41
       0.08       0.00       0.08       8696       0.23       1.41
Unrolled Double  Precision  8696 Kflops ; 10 Reps
bare_linpack_dp runs success!
```

注意：该示例当前仅支持在玄铁带双精度硬浮点的处理器上编译运行

3.2.4.5. bare_dhrystone

该示例位于solutions/bare_dhrystone，其用于评估处理器的速度和效率(整形计算)。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
Dhrystone Benchmark, Version 2.1 (Language: C)


Execution starts, 1000000 runs through Dhrystone
Execution ends

Final values of the variables used in the benchmark:

Int_Glob:            5
        should be:   5
Bool_Glob:           1
        should be:   1
Ch_1_Glob:           A
        should be:   A
Ch_2_Glob:           B
        should be:   B
Arr_1_Glob[8]:       7
        should be:   7
Arr_2_Glob[8][7]:    1000010
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          1612754832
        should be:   (implementation-dependent)
  Discr:             0
        should be:   0
  Enum_Comp:         2
        should be:   2
  Int_Comp:          17
        should be:   17
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->
  Ptr_Comp:          1612754832
        should be:   (implementation-dependent), same as above


```

3.2.5. 其他示例

3.2.5.1. bare_cpp_demo

该示例位于solutions/bare_cpp_demo，其是一个基于裸系统关于c++程序的使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
Object is being created
bare_cpp_demo demo start!
CPP DEMO Start!!!
Object is being created
Object is being created
Object is being created
Object is being created
Object is being created
Object is being created
g_box volume: 120.000000
box1 volume: 210.000000
box2 volume: 1560.000000

max lenth:7Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
CPP DEMO End!!!
```

注意：C++程序的支持需要默认增加DHAVE_INIT_ARRAY_LD编译选项，具体请参考对应解决方案下package.yaml编译配置文件中的用法

3.3. 面向mcu(使用原生RTOS接口)级别示例

玄铁RTOS SDK当前支持了freertos和rtthread两种RTOS类型，后续将加入更多RTOS类型支持。面向mcu级别的示例基于原生RTOS接口提供了最小系统，内核接口如事件、锁、信号量、消息队列等核心API的使用方法。

注意：

- 当前仅rtthread支持smp功能
- 面向mcu级别的示例暂不支持命令行(cli组件)，若项目中需要请自行参考面向soc级别的示例(支持命令行)移植相关代码。

3.3.1. freertos

3.3.1.1. mcu_freertos_helloworld

该示例位于solutions/mcu_freertos_helloworld，其是一个基于freertos原生接口的最小工程。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
FreeRTOS version:V10.4.3 LTS Patch 3
Hello world! FreeRTOS
Hello world! FreeRTOS
Hello world! FreeRTOS
```

3.3.1.2. 原生RTOS接口使用示例

3.3.1.2.1. mcu_freertos_event

该示例位于solutions/mcu_freertos_event，其是一个基于freertos原生的事件接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
FreeRTOS version:V10.4.3 LTS Patch 3
example_k_event writes event .
Example_Event waits event 0x1001
Example_Event,reads event :0x1001
test kernel event successfully!
```

3.3.1.2.2. mcu_freertos_message_q

该示例位于solutions/mcu_freertos_message_q，其是一个基于freertos原生的消息队列接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
FreeRTOS version:V10.4.3 LTS Patch 3
send_Entry
send_Entry:number of queued mesages : 1
send_Entry:number of queued mesages : 2
recv_Entry
send_Entry:number of queued mesages : 3
send_Entry:number of queued mesages : 4
send_Entry:number of queued mesages : 5
recv message:test is message 0
recv_Entry:number of queued mesages : 4
recv message:test is message 1
recv_Entry:number of queued mesages : 3
recv message:test is message 2
recv_Entry:number of queued mesages : 2
recv message:test is message 3
recv_Entry:number of queued mesages : 1
recv message:test is message 4
recv_Entry:number of queued mesages : 0
message queue recv finish
delete the queue successfully!
```

3.3.1.2.3. mcu_freertos_mutex

该示例位于solutions/mcu_freertos_mutex，其是一个基于freertos原生的互斥锁接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
FreeRTOS version:V10.4.3 LTS Patch 3
task2 try to get  mutex, wait forever.
task2 get mutex g_Testmux01 and suspend 100 ms.
task1 try to get  mutex, wait 100ms.
task2 resumed and post the g_Testmux01
task1 get mutex g_Testmux01.
test kernel mutex successfully!
```

3.3.1.2.4. mcu_freertos_sem

该示例位于solutions/mcu_freertos_sem，其是一个基于freertos原生的信号量接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
FreeRTOS version:V10.4.3 LTS Patch 3
Example_SemTask1 try get sem g_usSem ,timeout 100ms.
Example_SemTask1 post sem g_usSem .
Example_SemTask2 try get sem g_usSem wait forever.
Example_SemTask2 get sem g_usSem and then delay 200ms .
Example_SemTask2 post sem g_usSem .
test kernel semaphore successfully!
```

3.3.1.2.5. mcu_freertos_task

该示例位于solutions/mcu_freertos_task，其是一个基于freertos原生的多任务接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
FreeRTOS version:V10.4.3 LTS Patch 3
Enter TaskHi Handler.
Example_TaskLo:Enter TaskLo Handler.
Example_TaskHi:TaskHi vTaskDelay Done and suspend self.
Example_TaskLo:resume Example_TaskHi
Example_TaskHi:test kernel task successfully!
```

3.3.1.2.6. mcu_freertos_time

该示例位于solutions/mcu_freertos_time，其是一个基于freertos原生的时间相关接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
FreeRTOS version:V10.4.3 LTS Patch 3
xTaskGetTickCount = 0
print cnt every 1s for 10 times
-----9
-----8
-----7
-----6
-----5
-----4
-----3
-----2
-----1
-----0
print cnt every 3s for 3 times
-----2
-----1
-----0
xTaskGetTickCount = 2000
xTaskGetTickCount after delay = 2100
test kernel time successfully!
```

3.3.1.2.7. mcu_freertos_timer

该示例位于solutions/mcu_freertos_timer，其是一个基于freertos原生的定时器接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
FreeRTOS version:V10.4.3 LTS Patch 3
start to create timer
g_timercount1=1
tick_last1 = 50
g_timercount1=2
tick_last1 = 150
start Timer2
g_timercount2=1
tick_last2 = 210
g_timercount2=2
tick_last2 = 220
g_timercount2=3
tick_last2 = 230
g_timercount2=4
tick_last2 = 240
g_timercount2=5
tick_last2 = 250
g_timercount2=6
tick_last2 = 260
g_timercount2=7
tick_last2 = 270
g_timercount2=8
tick_last2 = 280
g_timercount2=9
tick_last2 = 290
g_timercount2=10
tick_last2 = 300
test kernel timer successfully!
```

3.3.1.3. 其他示例

3.3.1.3.1. mcu_freertos_cpp

该示例位于solutions/mcu_freertos_cpp，其是一个基于freertos系统关于c++程序的使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
Object is being created
FreeRTOS version:V10.4.3 LTS Patch 3
CPP DEMO Start!!!
Object is being created
Object is being created
Object is being created
Object is being created
Object is being created
Object is being created
g_box volume: 120.000000
box1 volume: 210.000000
box2 volume: 1560.000000

max lenth:7Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
CPP DEMO End!!!
```

注意：C++程序的支持需要默认增加DHAVE_INIT_ARRAY_LD编译选项，具体请参考对应解决方案下package.yaml编译配置文件中的用法

3.3.2. rtthread

3.3.2.1. mcu_rtthread_helloworld

该示例位于solutions/mcu_rtthread_helloworld，其是一个基于rtthread原生接口的最小工程。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Mar  7 2024 08:53:22
 2006 - 2022 Copyright by RT-Thread team
Hello world! RT-Thread
Hello world! RT-Thread

```

3.3.2.2. 原生RTOS接口使用示例

3.3.2.2.1. mcu_rtthread_event

该示例位于solutions/mcu_rtthread_event，其是一个基于rtthread原生的事件接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Mar 15 2024 08:02:56
 2006 - 2022 Copyright by RT-Thread team
thread2: send event3
thread1: OR recv event 0x8
thread1: delay 1s to prepare the second event
thread2: send event5
thread2: send event3
thread2 leave.
thread1: AND recv event 0x28
thread1 leave.
```

3.3.2.2.2. mcu_rtthread_message_q

该示例位于solutions/mcu_rtthread_message_q，其是一个基于rtthread原生的消息队列接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Mar 15 2024 08:03:51
 2006 - 2022 Copyright by RT-Thread team
thread2: send message - A
thread1: recv msg from msg queue, the content:A
thread2: send message - B
thread2: send message - C
thread2: send message - D
thread2: send message - E
thread2: send message - F
thread1: recv msg from msg queue, the content:B
thread2: send message - G
thread2: send message - H
thread2: send urgent message - I
thread2: send message - J
thread2: send message - K
thread1: recv msg from msg queue, the content:I
thread2: send message - L
thread2: send message - M
thread2: send message - N
thread2: send message - O
thread2: send message - P
thread1: recv msg from msg queue, the content:C
thread2: send message - Q
thread2: send message - R
thread2: send message - S
thread2: send message - T
message queue stop send, thread2 quit
thread1: recv msg from msg queue, the content:D
thread1: recv msg from msg queue, the content:E
thread1: recv msg from msg queue, the content:F
thread1: recv msg from msg queue, the content:G
thread1: recv msg from msg queue, the content:H
thread1: recv msg from msg queue, the content:J
thread1: recv msg from msg queue, the content:K
thread1: recv msg from msg queue, the content:L
thread1: recv msg from msg queue, the content:M
thread1: recv msg from msg queue, the content:N
thread1: recv msg from msg queue, the content:O
thread1: recv msg from msg queue, the content:P
thread1: recv msg from msg queue, the content:Q
thread1: recv msg from msg queue, the content:R
thread1: recv msg from msg queue, the content:S
```

```
45  thread1: recv msg from msg queue, the content:T
46  thread1: detach mq
```

3.3.2.2.3. mcu_rtthread_mutex

该示例位于solutions/mcu_rtthread_mutex，其是一个基于rtthread原生的互斥锁接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

Bash

```
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Mar 15 2024 08:05:18
 2006 - 2022 Copyright by RT-Thread team
rt_thread_entry2 mutex protect ,number1 = mumber2 is 1
rt_thread_entry1 mutex protect ,number1 = mumber2 is 2
rt_thread_entry2 mutex protect ,number1 = mumber2 is 3
rt_thread_entry1 mutex protect ,number1 = mumber2 is 4
rt_thread_entry2 mutex protect ,number1 = mumber2 is 5
rt_thread_entry1 mutex protect ,number1 = mumber2 is 6
rt_thread_entry2 mutex protect ,number1 = mumber2 is 7
rt_thread_entry1 mutex protect ,number1 = mumber2 is 8
rt_thread_entry2 mutex protect ,number1 = mumber2 is 9
rt_thread_entry2 exitx protect ,number1 = mumber2 is 10

rt_thread_entry1 mutex protect ,number1 = mumber2 is 11
rt_thread_entry1 mutex protect ,number1 = mumber2 is 12
rt_thread_entry1 mutex protect ,number1 = mumber2 is 13
rt_thread_entry1 mutex protect ,number1 = mumber2 is 14
rt_thread_entry1 mutex protect ,number1 = mumber2 is 15
rt_thread_entry1 mutex protect ,number1 = mumber2 is 16
rt_thread_entry1 mutex protect ,number1 = mumber2 is 17
rt_thread_entry1 mutex protect ,number1 = mumber2 is 18
rt_thread_entry1 mutex protect ,number1 = mumber2 is 19
rt_thread_entry1 mutex protect ,number1 = mumber2 is 20
rt_thread_entry1 exit

```

3.3.2.2.4. mcu_rtthread_sem

该示例位于solutions/mcu_rtthread_sem，其是一个基于rtthread原生的信号量接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Mar 15 2024 08:06:17
 2006 - 2022 Copyright by RT-Thread team
create done. dynamic semaphore value = 0.
thread1 release a dynamic semaphore.
thread1 release a dynamic semaphore.
thread1 release a dynamic semaphore.
thread1 release a dynamic semaphore.
thread1 release a dynamic semaphore.
thread1 release a dynamic semaphore.
thread1 release a dynamic semaphore.
thread1 release a dynamic semaphore.
thread1 release a dynamic semaphore.
thread1 release a dynamic semaphore.
rt_thread1_entry exit
thread2 take a dynamic semaphore. number = 1
thread2 take a dynamic semaphore. number = 2
thread2 take a dynamic semaphore. number = 3
thread2 take a dynamic semaphore. number = 4
thread2 take a dynamic semaphore. number = 5
thread2 take a dynamic semaphore. number = 6
thread2 take a dynamic semaphore. number = 7
thread2 take a dynamic semaphore. number = 8
thread2 take a dynamic semaphore. number = 9
thread2 take a dynamic semaphore. number = 10
rt_thread2_entry exit
```

3.3.2.2.5. mcu_rtthread_task

该示例位于solutions/mcu_rtthread_task，其是一个基于rtthread原生的任务接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Mar 15 2024 08:07:00
 2006 - 2022 Copyright by RT-Thread team
thread1 count: 0
thread2 count: 0
thread2 count: 1
thread2 count: 2
thread2 count: 3
thread2 count: 4
thread2 count: 5
thread2 count: 6
thread2 count: 7
thread2 count: 8
thread2 count: 9
thread2 exit
thread1 count: 1
thread1 count: 2
thread1 count: 3
thread1 count: 4
thread1 exit
```

3.3.2.2.6. mcu_rtthread_time

该示例位于solutions/mcu_rtthread_time，其是一个基于rtthread原生的时间相关接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Mar 15 2024 08:07:41
 2006 - 2022 Copyright by RT-Thread team
current  kernel systick 10 ms
current tick count = 1,  will delay 19s....
print cnt every 1s for 10 times
-----9
-----8
-----7
-----6
-----5
-----4
-----3
-----2
-----1
-----0
print cnt every 3s for 3 times
-----2
-----1
-----0
tick cont = 1902 after 19s, will delay 1s....
tick cont = 2002 after delay 1s
rtt time successfully!
```

3.3.2.2.7. mcu_rtthread_timer

该示例位于solutions/mcu_rtthread_timer，其是一个基于rtthread原生的定时器接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Mar 15 2024 08:10:48
 2006 - 2022 Copyright by RT-Thread team
periodic timer is timeout 0
periodic timer is timeout 1
one shot timer is timeout
periodic timer is timeout 2
periodic timer is timeout 3
periodic timer is timeout 4
periodic timer is timeout 5
periodic timer is timeout 6
periodic timer is timeout 7
periodic timer is timeout 8
periodic timer is timeout 9
periodic timer was stopped!
```

3.3.2.3. 多核示例

3.3.2.3.1. mcu_rtthread_smp

该示例位于solutions/mcu_rtthread_smp，其是一个基于rtthread原生接口的最小SMP工程(默认2 core)。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Apr  1 2024 07:56:18
 2006 - 2022 Copyright by RT-Thread team
[thread_0] in 0 core, count:0
[thread_1] in 1 core, count:1
[thread_0] in 0 core, count:2
[thread_1] in 1 core, count:3
[thread_0] in 0 core, count:4
[thread_1] in 1 core, count:5
[thread_0] in 0 core, count:6
[thread_1] in 1 core, count:7
[thread_0] in 0 core, count:8
[thread_1] in 1 core, count:9
[thread_0] in 0 core, count:10
[thread_1] in 1 core, count:11
[thread_0] in 0 core, count:12
[thread_1] in 1 core, count:13
```

smp功能的使能主要依赖下面这两个配置(参考solutions/mcu_rtthread_smp/package.yaml):

```Bash
    # SMP CONFIG


    CONFIG_SMP: 1
    CONFIG_NR_CPUS: 2
```

3.3.2.4. 其他示例

3.3.2.4.1. mcu_rtthread_cpp

该示例位于solutions/，其是一个基于rtthread原生接口的最小工程。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下:

```
 \ | /
- RT -     Thread Operating System
 / | \     5.0.1 build Apr  1 2024 07:59:06
 2006 - 2022 Copyright by RT-Thread team
Object is being created
CPP DEMO Start!!!
Object is being created
Object is being created
Object is being created
Object is being created
Object is being created
Object is being created
g_box volume: 120.000000
box1 volume: 210.000000
box2 volume: 1560.000000

max lenth:7Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
CPP DEMO End!!!
```

3.4. 面向soc(使用OSAL接口)级别示例

3.4.1. soc_helloworld

该示例位于solutions/，其是一个基于OSAL接口的最小工程。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar  6 2024,02:53:39]
[   0.000]<I>[INIT]<app_task>Build:Mar  6 2024,02:53:39
[   0.010]<I>[app]<app_task>kernel version : FreeRTOS V10.4.3 LTS Pa

[   0.020]<D>[app]<app_task>Hello world! RTOS SDK
(cli-uart)# [   3.020]<D>[app]<app_task>Hello world! RTOS SDK
[   6.020]<D>[app]<app_task>Hello world! RTOS SDK
[   9.020]<D>[app]<app_task>Hello world! RTOS SDK
```

3.4.2. 内核示例

3.4.2.1. soc_kernel_event

该示例位于solutions/soc_kernel_event，其是一个基于OSAL封装的事件接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:27:34]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:27:34
[   0.010]<I>[app]<app_task>kernel version : FreeRTOS V10.4.3 LTS Patch 3

[   0.020]<I>[soc_event_test]<app_task>example_k_event writes event .
[   0.020]<I>[soc_event_test]<example_event_task>Example_Event waits even
t 0x1001
[   0.030]<I>[soc_event_test]<example_event_task>reads event :0x1001.
[   0.040]<I>[soc_event_test]<example_event_task>test kernel event success
fully!
(cli-uart)#
```

3.4.2.2. soc_kernel_message_q

该示例位于solutions/soc_kernel_message_q，其是一个基于OSAL封装的消息队列接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:28:22]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:28:22
[   0.010]<I>[app]<app_task>kernel version : FreeRTOS V10.4.3 LTS Patch 3

[   0.020]<I>[soc_message_q]<send_task>send_entry start .
[   0.020]<I>[soc_message_q]<send_task>send_entry:number of queued mesages : 1 .
[   0.030]<I>[soc_message_q]<recv_task>send_entry start .
(cli-uart)# [   0.040]<I>[soc_message_q]<send_task>send_entry:number of queued mesages : 2 .
[   0.050]<I>[soc_message_q]<send_task>send_entry:number of queued mesages : 3 .
[   0.060]<I>[soc_message_q]<send_task>send_entry:number of queued mesages : 4 .
[   0.070]<I>[soc_message_q]<send_task>send_entry:number of queued mesages : 5 .
[   0.130]<I>[soc_message_q]<recv_task>recv message:test is message 0, recv message size:64

[   0.130]<I>[soc_message_q]<recv_task>number of queued mesages : 4 .
[   0.150]<I>[soc_message_q]<recv_task>recv message:test is message 1, recv message size:64

[   0.150]<I>[soc_message_q]<recv_task>number of queued mesages : 3 .
[   0.170]<I>[soc_message_q]<recv_task>recv message:test is message 2, recv message size:64

[   0.170]<I>[soc_message_q]<recv_task>number of queued mesages : 2 .
[   0.190]<I>[soc_message_q]<recv_task>recv message:test is message 3, recv message size:64

[   0.190]<I>[soc_message_q]<recv_task>number of queued mesages : 1 .
[   0.210]<I>[soc_message_q]<recv_task>recv message:test is message 4, recv message size:64

[   0.210]<I>[soc_message_q]<recv_task>number of queued mesages : 0 .
[   0.330]<I>[soc_message_q]<recv_task>queue recv finish .
```

3.4.2.3. soc_kernel_mutex

该示例位于solutions/soc_kernel_mutex，其是一个基于OSAL封装的互斥锁接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:29:15]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:29:15
[   0.010]<I>[app]<app_task>kernel version : FreeRTOS V10.4.3 LTS Patch 3

[   0.020]<I>[soc_mutex_test]<mutex_task1>mutex_task2 lock
[   0.020]<I>[soc_mutex_test]<mutex_task1>delay 200ms
[   0.030]<I>[soc_mutex_test]<mutex_task1>delay 10ms
(cli-uart)# [   0.040]<I>[soc_mutex_test]<mutex_task1>task1 lock
[   0.140]<I>[soc_mutex_test]<mutex_task1>timeout and try to get  mutex, wait forever
[   0.230]<I>[soc_mutex_test]<mutex_task1>mutex_task2 unlock
[   0.230]<I>[soc_mutex_test]<mutex_task1>task1 unlock
```

3.4.2.4. soc_kernel_sem

该示例位于solutions/soc_kernel_sem，其是一个基于OSAL封装的信号量接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:29:47]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:29:47
[   0.010]<I>[app]<app_task>kernel version : FreeRTOS V10.4.3 LTS Patch 3

[   0.020]<I>[soc_sem_test]<task_hi task>task_hi_task start,get sem wait forever
[   0.020]<I>[soc_sem_test]<task_hi task>sem wait
[   0.030]<I>[soc_sem_test]<task_hi task>will delay 200ms
[   0.030]<I>[soc_sem_test]<task_lo task>task_lo_task start, try to wait 100ms
(cli-uart)# [   0.150]<I>[soc_sem_test]<task_lo task>aos_sem_wait timeout, will wait forever
[   0.230]<I>[soc_sem_test]<task_hi task>sem signal
[   0.230]<I>[soc_sem_test]<task_lo task>sem signal
```

3.4.2.5. soc_kernel_task

该示例位于solutions/soc_kernel_task，其是一个基于OSAL封装的多任务接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:31:04]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:31:04
[   0.010]<I>[app]<app_task>kernel version : FreeRTOS V10.4.3 LTS Patch 3

[   0.020]<I>[soc_task_test]<task_hi task>task_hi_task start
[   0.020]<I>[soc_task_test]<task_lo task>task_lo_task start
[   0.030]<I>[soc_task_test]<task_hi task>task_hi_task delay end,start to suspend self
(cli-uart)# [   0.080]<I>[soc_task_test]<task_hi task>suspend self success ful
[   0.080]<I>[soc_task_test]<task_lo task>resume  task_hi_task success
```

3.4.2.6. soc_kernel_time

该示例位于solutions/soc_kernel_time，其是一个基于OSAL封装的时间相关接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:31:42]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:31:42
[   0.010]<I>[app]<app_task>kernel version : FreeRTOS V10.4.3 LTS Patch 3

kernel systick is 10 ms
current tick count = 2 before 19s....
print cnt every 1s for 10 times
(cli-uart)# -----9
-----8
-----7
-----6
-----5
-----4
-----3
-----2
-----1
-----0
print cnt every 3s for 3 times
-----2
-----1
-----0
tick count = 1903 after 19s, will be delay  1s....
tick count = 2003 after delay 1s
test kernel time successfully!
```

3.4.2.7. soc_kernel_timer

该示例位于solutions/soc_kernel_timer，其是一个基于OSAL封装的定时器接口使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:44:38]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:44:38
[   0.010]<I>[app]<app_task>kernel version : FreeRTOS V10.4.3 LTS Patch 3

[   0.020]<I>[soc_timer_test]<app_task>timer1 start
[   0.020]<I>[soc_timer_test]<app_task>delay 1000 ms
(cli-uart)# [   1.030]<I>[soc_timer_test]<app_task>timer1 stop
[   1.030]<I>[soc_timer_test]<app_task>timer1 call success
[   1.040]<I>[soc_timer_test]<app_task>timer2 start
[   1.040]<I>[soc_timer_test]<app_task>delay 1010 ms
[   2.050]<I>[soc_timer_test]<app_task>timer2 stop
[   2.050]<I>[soc_timer_test]<app_task>every 100ms call once
[   2.060]<I>[soc_timer_test]<app_task>the 0 trigger time of the timer is
 1140 ms
[   2.060]<I>[soc_timer_test]<app_task>the 1 trigger time of the timer is
 1240 ms
[   2.070]<I>[soc_timer_test]<app_task>the 2 trigger time of the timer is
 1340 ms
[   2.080]<I>[soc_timer_test]<app_task>the 3 trigger time of the timer is
 1440 ms
[   2.090]<I>[soc_timer_test]<app_task>the 4 trigger time of the timer is
 1540 ms
[   2.090]<I>[soc_timer_test]<app_task>the 5 trigger time of the timer is
 1640 ms
[   2.100]<I>[soc_timer_test]<app_task>the 6 trigger time of the timer is
 1740 ms
[   2.110]<I>[soc_timer_test]<app_task>the 7 trigger time of the timer is
 1840 ms
[   2.120]<I>[soc_timer_test]<app_task>the 8 trigger time of the timer is
 1940 ms
[   2.120]<I>[soc_timer_test]<app_task>the 9 trigger time of the timer is
 2040 ms
[   2.130]<I>[soc_timer_test]<app_task>timer test success
```

3.4.3. cpu core相关示例

3.4.3.1. soc_core_dsp

该示例位于solutions/soc_core_dsp，其是一个面向soc领域(使用osal封装)的玄铁cpu dsp扩展多线程使用示例，用于演示玄铁处理器P扩展使用方式。示例程序中会创建多个线程调用libcsi_xt900p32fd_dsp.a(采用P扩展指令封装，实际存储在components/csi/csi2/dsp文件夹中)库中的csi_abs_f32函数计算。

基于玄铁内部smartl FPGA(以e907fdp为例)平台上的默认运行结果如下：

```Bash
###Welcome to RTOS SDK, based XuanTie e907fdp###
[Mar 11 2024,07:41:15]
[   0.010]<I>[INIT]<app_task>Build:Mar 11 2024,07:41:15
soc_core_dsp demo start!
(cli-uart)# dsp runs successfully!
```

注意：该示例仅支持玄铁处理器带P扩展的型号上编译运行

3.4.3.2. soc_core_vector

该示例位于solutions/soc_core_vector，其是一个面向soc领域(使用osal封装)的玄铁cpu vector扩展多线程使用示例，用于演示玄铁处理器V扩展使用方式。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:48:44]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:48:44
soc_core_vector demo start!
[fp32] gemm start
[fp32] gemm start
=====>>native tes
=====>>native test start.
=====>>native tes
=====>>native tes
=====>>native test start.
=====>>native test start.
=====>>native test start.
=====>>native test end.

=====>>native test end.
=====>>vector test start.
===vlen=256
===vlen=256
           e test end.
=====>>vector test start.
===vlen=256
===vlen=256
           e test end.
=====>>vector test start.
===vlen=256
===vlen=256
           r test start.=====>>vector test start.
===vlen=256
(cli-uart)# =====>>native test end.
=====>>vector test start.
===vlen=256
=====>>vector test end.
native time = 630ms, vector time = 410ms. diff = 220ms
[fp32] gemm end, total diff time = 1300ms
[fp32] gemm end, total dinative time = 410ms, vector time = 420ms. diff =
4294967286ms
[fp32] gemm end, total diff time = 1250ms
[fp32] gemm end, total dinative time = 380ms, vector time = 420ms. diff =
4294967256ms
[fp32] gemm end, total diff time = 1200ms
[fp32] gemm end, total dinative time = 400ms, vector time = 440ms. diff =
4294967256ms
[fp32] gemm end, total diff time = 1130ms
```

```
=====>>vector test end.
native time = 240ms, vector time = 400ms. diff = 4294967136ms
[fp32] gemm end, total diff time = 1080ms
vector runs successfully!
```

注意：该示例仅支持玄铁处理器带v扩展的型号上编译运行

3.4.3.3. soc_core_matrix

该示例位于solutions/soc_core_matrix，其是一个面向soc领域(使用osal封装)的玄铁cpu matrix扩展多线程使用示例，用于演示玄铁处理器matrix扩展使用方式。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

Bash

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:51:34]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:51:34
soc_core_matrix demo start!
[int8] gemm start
[int8] gemm start
[int8] gemm start
[int8] gemm start
(cli-uart)# [m * k * n = 48 * 48 * 48] best execution time: 70.000ms, 0.00 GFLOPS
[m * k * n = 48 * 48 * 48] best execution time: 70.000ms, 0.00 GFLOPS
[m * k * n = 48 * 48 * 48] best execution time: 70.000ms, 0.00 GFLOPS
[m * k * n = 48 * 48 * 48] best execution time: 70.000ms, 0.00 GFLOPS
[m * k * n = 96 * 96 * 96] best execution time: 40.000ms, 0.04 GFLOPS
[int8] gemm end
[int8] gemm end
                * 96 * 96] best execution time: 50.000ms, 0.04 GFLOPS
[int8] gemm end
[int8] gemm end
                * 96 * 96] best execution time: 40.000ms, 0.04 GFLOPS
[int8] gemm end
[m * k * n = 96 * 96 * 96] best execution time: 40.000ms, 0.04 GFLOPS
[int8] gemm end
matrix runs successfully!
```

注意：该示例仅支持玄铁处理器带matrix扩展的型号上编译运行

3.4.4. 多核示例

3.4.4.1. soc_smp_demo

该示例位于solutions/soc_smp_demo，其是一个基于OSAL接口的最小SMP工程(默认2 core，当前仅rtthread支持多核)。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```Bash
  ###Welcome to YoC, based XuanTie c907fdvm###
[Apr  1 2024,07:59:57]
[   0.010]<I>[INIT]<app_task>Build:Apr  1 2024,07:59:57
(cli-uart)# [thread_1] in 1 core, count:0
[thread_0] in 0 core, count:1
[thread_1] in 1 core, count:2
[thread_0] in 0 core, count:3
[thread_1] in 1 core, count:4
[thread_0] in 0 core, count:5
[thread_1] in 1 core, count:6
[thread_0] in 0 core, count:7
[thread_1] in 1 core, count:8
[thread_0] in 0 core, count:9
```

smp功能的使能主要依赖下面这两个配置(参考solutions/soc_smp_demo/package.yaml)：

```Bash
    # SMP CONFIG


    CONFIG_SMP: 1
    CONFIG_NR_CPUS: 2
```

3.4.5. 其他示例

3.4.5.1. soc_cpp_demo

该示例位于solutions/soc_cpp_demo，其是一个基于OSAL接口的C++使用示例。

基于玄铁内部xiaohui FPGA(以c907fdvm为例)平台上的默认运行结果如下：

```
###Welcome to RTOS SDK, based XuanTie c907fdvm###
[Mar 15 2024,08:53:31]
[   0.000]<I>[INIT]<app_task>Build:Mar 15 2024,08:53:31
Object is being created
[   0.010]<I>[app]<app_task>kernel version : FreeRTOS V10.4.3 LTS Patch 3

CPP DEMO Start!!!
Object is being created
Object is being created
Object is being created
Object is being created
Object is being created
Object is being created
g_box volume: 120.000000
box1 volume: 210.000000
box2 volume: 1560.000000
max lenth:7
Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
Object is being deleted
CPP DEMO End!!!
(cli-uart)#
```

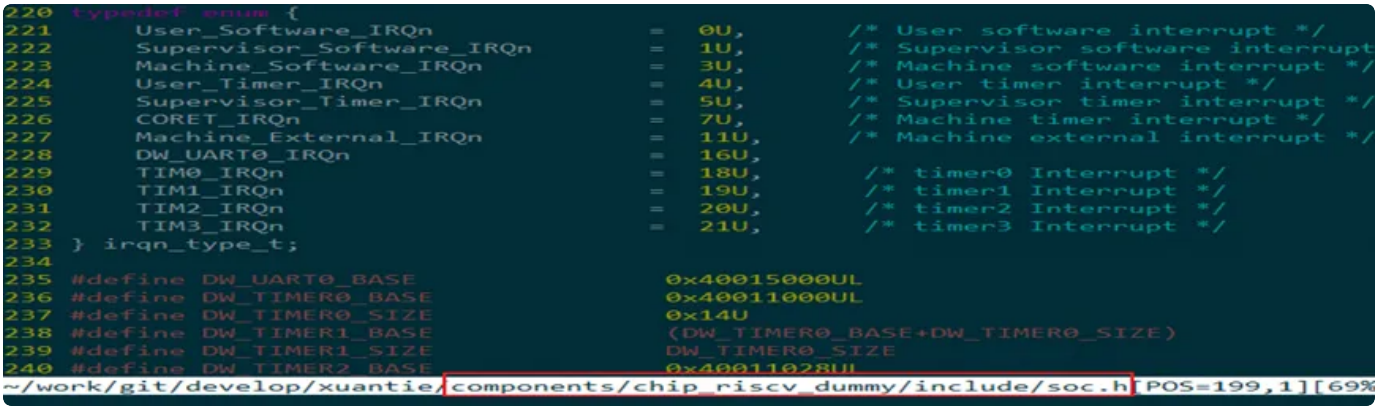
4. 移植说明

由于客户是基于玄铁处理器RTL相关代码进行SOC设计，地址空间/外设/中断号等跟玄铁RTOS SDK中默认定义大概率有差异，因此客户需要根据实际SOC设计修改相关代码才能成功运行SDK中提供的各种示例。玄铁内部的FPGA验证平台主要有两个，分别为针对E9xx系列的smartl和针对C/R9xx系列的xiaohui平台。这些FPGA平台并不会直接提供给用户。玄铁内部E9xx和C/R9xx处理器由于编程模型不太相同，制作对应的bit时在地址空间、中断、时钟等方面也不同。用户在基于玄铁处理器开发SOC时请参考对应平台的代码移植，具体说明如下。

4.1. 地址空间和中断

地址空间和中断号的定义请参考components/chip_riscv_dummy/include/soc.h文件。当前SDK中仅提供了uart、timer(不同于clint中的系统定时器)两种外设的使用参考，用户可参考相关驱动代码按照

CSI2.0中定义的接口标准实现。

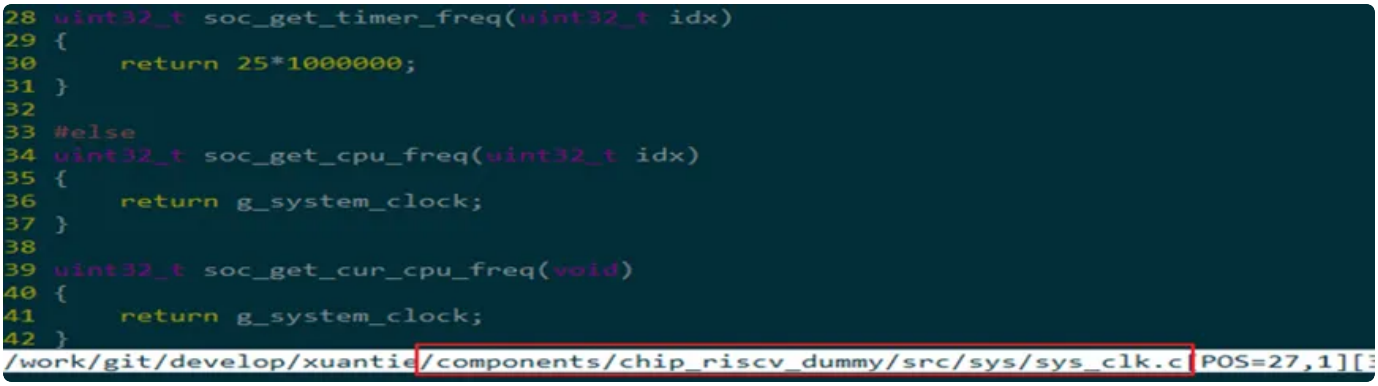


其中，

- 中断和异常处理函数统一定义在chip组件中的vectors.S文件中
- 异常栈和中断栈是独立的，统一由CONFIG_ARCH_INTERRUPTSTACK配置
- E9xx和C/R9xx默认基于M态(机器模式)运行
- E9xx和C/R9xx默认的中断栈大小分别为4KB/8KB，均由CONFIG_ARCH_INTERRUPTSTACK配置
- E9xx默认支持中断嵌套功能，由CONFIG_SUPPORT_IRQ_NESTED配置
- E9xx中断和异常入口函数分别为Default_IRQHandler/Default_Handler
- E9xx支持不可屏蔽中断(NMI)，异常号为24，实现在Default_Handler。NMI中断在smartl参考soc设计中绑定到DW_TIMER4_BASE定时器
- C/R9xx中分CLINT(处理器核局部中断控制器)中断和PLIC(平台级中断控制器)中断
- C/R9xx中机器模式软件/定时器/外部中断入口函数分别为Mtspend_Handler/Mcoret_Handler/Mirq_Handler
- C/R9xx需要在配置文件中配置CONFIG_PLIC_BASE和CONFIG_VIC_TSPDR两个宏定义的地址，分别指代中断控制器PLIC和处理器核局部中断CLINT的基地址，否则中断和系统调度可能出现异常。

4.2. 时钟

时钟配置相关代码请参考SDK中components/chip_riscv_dummy/src/sys/sys_clk.c文件，主要提供cpu 、coretimer(clint中的定时器，作为系统定时器)、uart、外设timer等时钟的配置，用户可根据实际SOC设计参考修改。

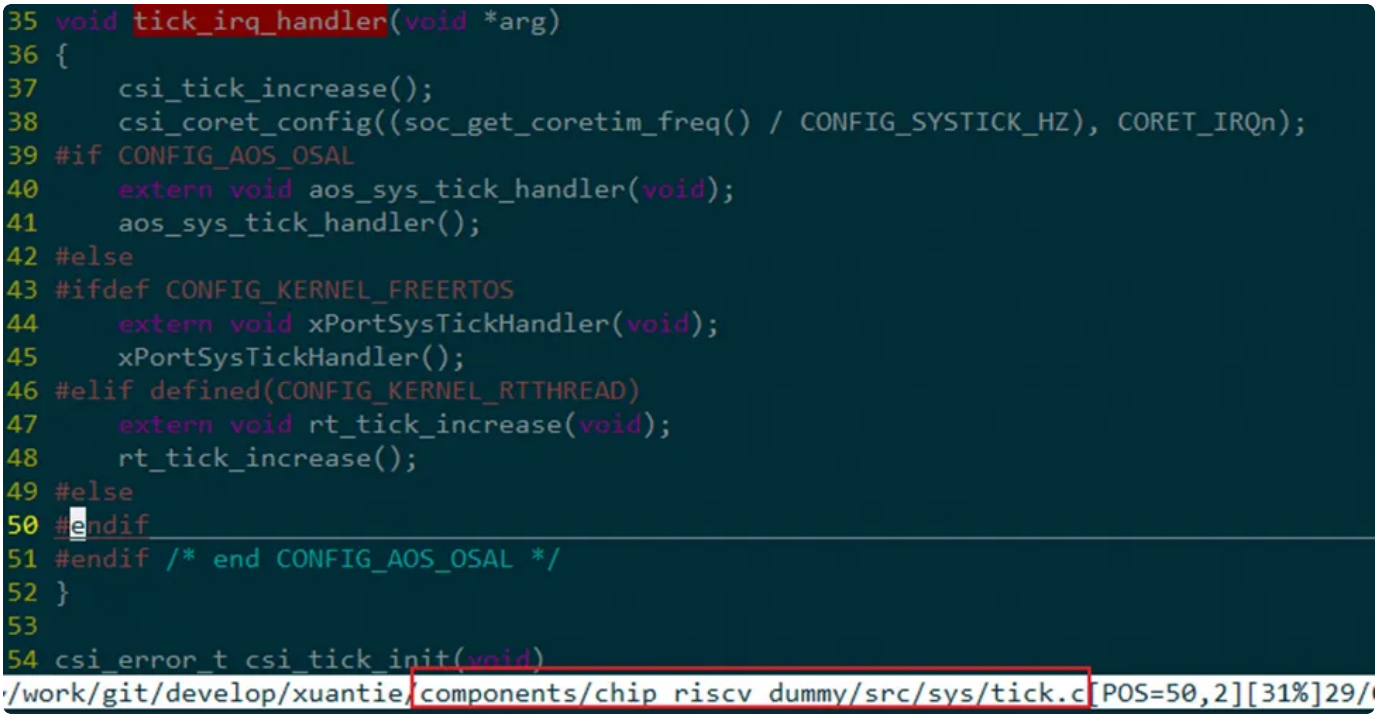


其中：

- 对于E9xx，这些时钟的默认均为20M
- 对于C/R9xx，cpu 、coretimer、uart、外设timer等时钟的配置，分别为50M、25M、36M、25M

4.3. 系统滴答和计时

系统滴答和计时(mdelay等时间函数)实现在components/chip_riscv_dummy/src/sys/tick.c，在玄铁RV处理器中采用CLINT(处理器核局部中断)中的计时器中断实现。



其中，

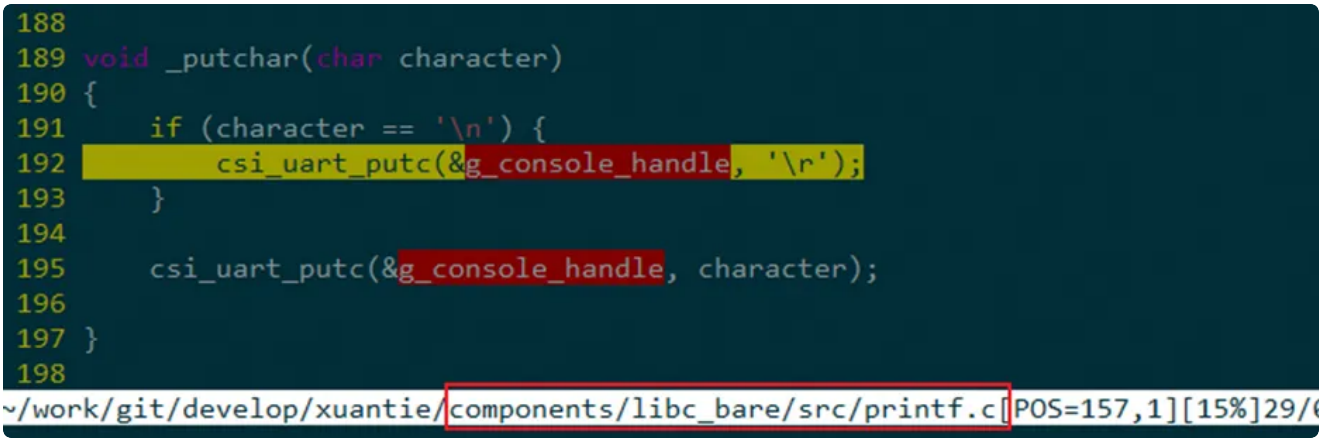
- 获取系统时间和RTOS系统调度均依赖于此
- 默认的系统滴答为100，由CONFIG_SYSTICK_HZ宏定义

4.4. 串口输出

在移植时，printf函数至关重要。在关注所采用uart地址空间、中断号的同时，还需要根据CSI2.0接口实现uart相关接口。

```
25 #if CONFIG_DEVICES_RVM_HAL
26 void board_uart_init(void)
27 {
28     rvm_uart_drv_register(0);
29 }
30 #else
31 __attribute__((weak)) csi_uart_t g_console_handle;
32
33 void board_uart_init(void)
34 {
35     /* init the console */
36     csi_uart_init(&g_console_handle, CONSOLE_UART_IDX);
37
38     /* config the UART */
39     csi_uart_baud(&g_console_handle, CONFIG_CLI_USART_BAUD);
40     csi_uart_format(&g_console_handle, UART_DATA_BITS_8, UART_PARITY_NONE, UART_
41 }
42 #endif
~/work/git/develop/xuantie/boards/board_riscv_dummy/src/uart/board_uart.c[POS=15,1][
```

```
193 _ssize_t _write_r(struct _reent *ptr, int fd, const void *buf, size_t nbytes)
194 {
195     int ret = -1;
196
197     if (buf == NULL) {
198         return 0;
199     }
200
201     if ((fd >= FD_VFS_START) && (fd <= FD_VFS_END)) {
202 #ifdef AOS_COMP_VFS
203         ret = aos_write(fd, buf, nbytes);
204 #endif
205         if (ret < 0) {
206             ret = -1;
207         }
208         return ret;
209 #if defined(CONFIG_TCPIP) || defined(CONFIG_SAL)
210     } else if ((fd >= FD_SOCKET_START) && (fd <= FD_EVENT_END)) {
211 #if defined(CONFIG_SAL)
212         return send(fd, buf, nbytes, 0);
213 #else
214         return lwip_write(fd, buf, nbytes);
215 #endif
216 #endif
217     } else if ((fd == STDOUT_FILENO) || (fd == STDERR_FILENO)) {
218         extern int uart_write(const void *buf, size_t size);
219         uart_write(buf, nbytes);
220         return nbytes;
221     } else {
222         return -1;
223     }
224 }
225
~/work/git/develop/xuantie/components/libc_stub/newlib_stub.c[POS=184,12][32%]29/03/24 -1
```

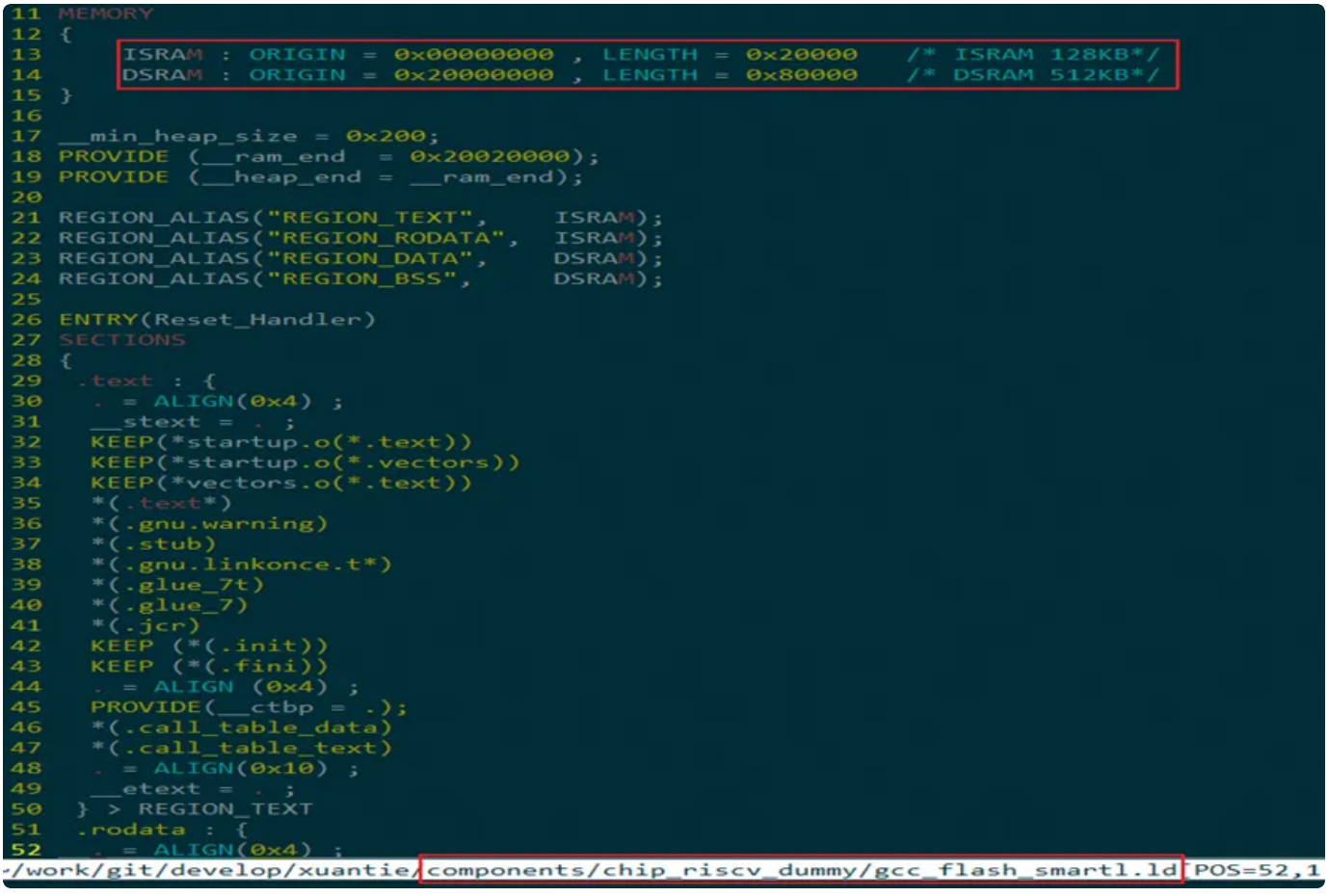


其中，

- 对于面向裸系统和mcu领域的示例，printf默认直接对接到csi_uart_xxx接口上，并使用loop模式
- 对于面向soc领域的示例，printf默认对接到rvm_hal设备驱动框架层，采用中断模式

4.5. 链接脚本

4.5.1. E9xx



- 链接脚本请参考components/chip_riscv_dummy/gcc_flash_smartl.ld文件
- 默认ISRAM(0x00000000~0x00020000)，DSRAM(0x20000000~0x20080000)
- 默认支持XIP模式执行

4.5.2. C/R9xx

```
MEMORY
{
    DRAM    : ORIGIN = 0x50000000, LENGTH = 0x100000    /* on-chip DRAM 1*1MB */
}

__min_heap_size = 0x200;
PROVIDE (__ram_end   = 0x50100000 - 0x8);
PROVIDE (__heap_end = __ram_end);

REGION_ALIAS("REGION_TEXT",     DRAM);
REGION_ALIAS("REGION_RODATA",   DRAM);
REGION_ALIAS("REGION_DATA",     DRAM);
REGION_ALIAS("REGION_BSS",      DRAM);

ENTRY(Reset_Handler)
SECTIONS
{
  .text : {
    . = ALIGN(0x4) ;
    __stext = . ;
    KEEP(*startup.o(*.text))
    KEEP(*startup.o(*.vectors))
    KEEP(*vectors.o(*.text))
    *(.text)
    *(.text*)
    *(.text.*)
    *(.gnu.warning)
    *(.stub)
    *(.gnu.linkonce.t*)
    *(.glue_7t)
    *(.glue_7)
    *(.jcr)
    KEEP (*(.init))
    KEEP (*(.fini))
    . = ALIGN(0x4) ;
    PROVIDE(__ctbp = .);
    *(.call_table_data)
    *(.call_table_text)
    . = ALIGN(0x4) ;
/work/git/develop/xuantie/components/chip_riscv_dummy/gcc_flash_xiaohui.ld[POS=43,1
```

- 链接脚本参考components/chip_riscv_dummy/gcc_flash_xiaohui.ld
- 默认DRAM(0x50000000~0x50100000)
- 默认不支持XIP执行

4.6. RTOS移植

4.6.1. FreeRTOS

FreeRTOS当前仅支持单核，在移植时除4.3节中系统滴答外，还需要关注如下移植点：

- 任务栈初始化

- 使能/关闭全局中断
- 进入/退出临界区
- 系统调度函数(vPortYield)
- 软件中断处理函数(tspend_handler)

重要的移植接口请参考下表：

主要函数	说明
StackType_t* pxPortInitialiseStack(StackType_t *pxTopOfStack, TaskFunction_t pxCode, void *pvParameters)	任务栈初始化，包含通用寄存器，浮点/向量/matrix寄存器(若需要)等。在创建任务时会被调用
void vPortEnableInterrupt(void)	使能全局中断
void vPortDisableInterrupt(void)	关闭全局中断
void vPortEnterCritical(void)	进入临界区
void vPortExitCritical(void)	退出临界区
void vPortYield(void)	用于系统调度(任务切换)，在RISCV处理器中通过配置CLINT中的软件中断相关寄存器触发调度
void xPortSysTickHandler(void)	系统滴答函数，在处理器定时器(coret)中断中被调用
void tspend_handler (void)	软件中断处理函数(汇编实现)，在此函数中实现任务切换

以E907FDP为例，

```
.global cpu_intrpt_save
.type cpu_intrpt_save, %function
cpu_intrpt_save:
    csrr    a0, mstatus
    csrc    mstatus, 8
    ret

.global cpu_intrpt_restore
.type cpu_intrpt_restore, %function
cpu_intrpt_restore:
    csrw    mstatus, a0
    ret

.global cpu_is_irq_enable
.type cpu_is_irq_enable, %function
cpu_is_irq_enable:
    csrr a0, mstatus
    andi a0, a0, RISCV_MSTATUS_MIE
    ret

/*************************************************************
 * Functions: vPortYield
 *
 *************************************************************/
.global vPortYield
.type   vPortYield, %function
vPortYield:
    li      t0, 0xE080100C
    lb      t1, (t0)
    li      t2, 0x01
    or      t1, t1, t2
    sb      t1, (t0)

    /* FIXME: trriger soft-irq quickly */
    fence
    nop
    nop
    nop
    nop
    nop

    ret
~/work/git/develop/xuantie/components/freertos/FreeRTOS/Source/portable/GCC/riscv/e907fdp/tspend/cpu_task_sw.S
```

```
StackType_t *pxPortInitialiseStack( StackType_t *pxTopOfStack, TaskFunction_t pxCode, void *pvParameters )
#endif
{
    StackType_t *stk  = NULL;
    register int *gp asm("x3");
    uint32_t temp = (uint32_t)pxTopOfStack;

    temp &= 0xFFFFFFF8UL;
    stk = (StackType_t *)temp;
    *(--stk)  = (uint32_t)pxCode;            /* Entry Point */
    *(--stk)  = (uint32_t)0x31313131L;       /* X31         */
    *(--stk)  = (uint32_t)0x30303030L;       /* X30         */
    *(--stk)  = (uint32_t)0x29292929L;       /* X29         */
    *(--stk)  = (uint32_t)0x28282828L;       /* X28         */
    *(--stk)  = (uint32_t)0x27272727L;       /* X27         */
    *(--stk)  = (uint32_t)0x26262626L;       /* X26         */
    *(--stk)  = (uint32_t)0x25252525L;       /* X25         */
    *(--stk)  = (uint32_t)0x24242424L;       /* X24         */
    *(--stk)  = (uint32_t)0x23232323L;       /* X23         */
    *(--stk)  = (uint32_t)0x22222222L;       /* X22         */
    *(--stk)  = (uint32_t)0x21212121L;       /* X21         */
    *(--stk)  = (uint32_t)0x20202020L;       /* X20         */
    *(--stk)  = (uint32_t)0x19191919L;       /* X19         */
    *(--stk)  = (uint32_t)0x18181818L;       /* X18         */
    *(--stk)  = (uint32_t)0x17171717L;       /* X17         */
    *(--stk)  = (uint32_t)0x16161616L;       /* X16         */
    *(--stk)  = (uint32_t)0x15151515L;       /* X15         */
    *(--stk)  = (uint32_t)0x14141414L;       /* X14         */
    *(--stk)  = (uint32_t)0x13131313L;       /* X13         */
    *(--stk)  = (uint32_t)0x12121212L;       /* X12         */
    *(--stk)  = (uint32_t)0x11111111L;       /* X11         */
    *(--stk)  = (uint32_t)pvParameters;      /* X10         */
    *(--stk)  = (uint32_t)0x09090909L;       /* X9          */
    *(--stk)  = (uint32_t)0x08080808L;       /* X8          */
    *(--stk)  = (uint32_t)0x07070707L;       /* X7          */
    *(--stk)  = (uint32_t)0x06060606L;       /* X6          */
    *(--stk)  = (uint32_t)0x05050505L;       /* X5          */
    *(--stk)  = (uint32_t)0x04040404L;       /* X4          */
    *(--stk)  = (uint32_t)gp;                /* X3          */
#if CONFIG_AOS_OSAL
    *(--stk)  = (uint32_t)aos_task_exit;     /* X1          */
~/work/git/develop/xuantie/components/freertos/FreeRTOS/Source/portable/GCC/riscv/e907fdp/port.c[POS=65,1][36%]2
```

相关移植代码路径如下：

- 任务栈初始化等：components/freertos/FreeRTOS/Source/portable/GCC/riscv/e907fdp/port.c
- 系统调度：components/freertos/FreeRTOS/Source/portable/GCC/riscv/e907fdp/tspend/cpu_task_sw.S
- 移植宏定义：components/freertos/FreeRTOS/Source/portable/GCC/riscv/e907fdp/tspend/portmacro.h
- 板级配置头文件：boards/board_riscv_dummy/include/FreeRTOSConfig.h

4.6.2. RTThread

RTThread当前支持SMP功能，在移植时除4.3节中系统滴答外，还需要关注如下移植点：

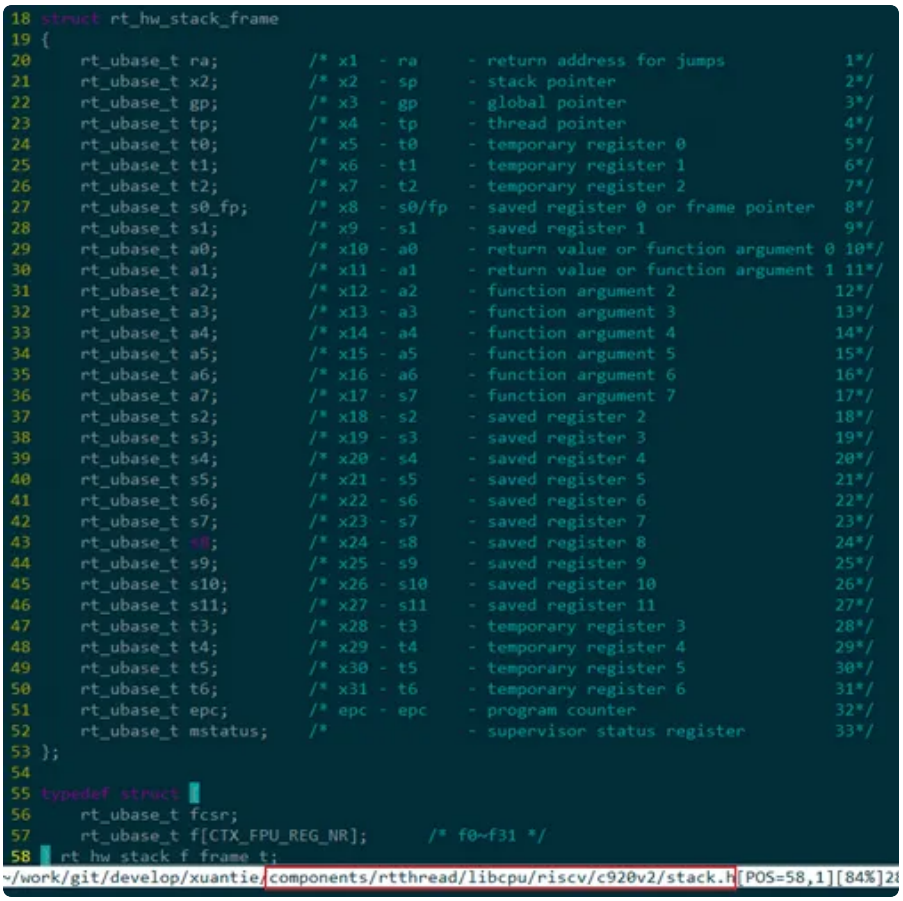
- 任务栈初始化
- 使能/关闭全局中断
- 系统调度函数
- IPI核间中断函数(SMP)

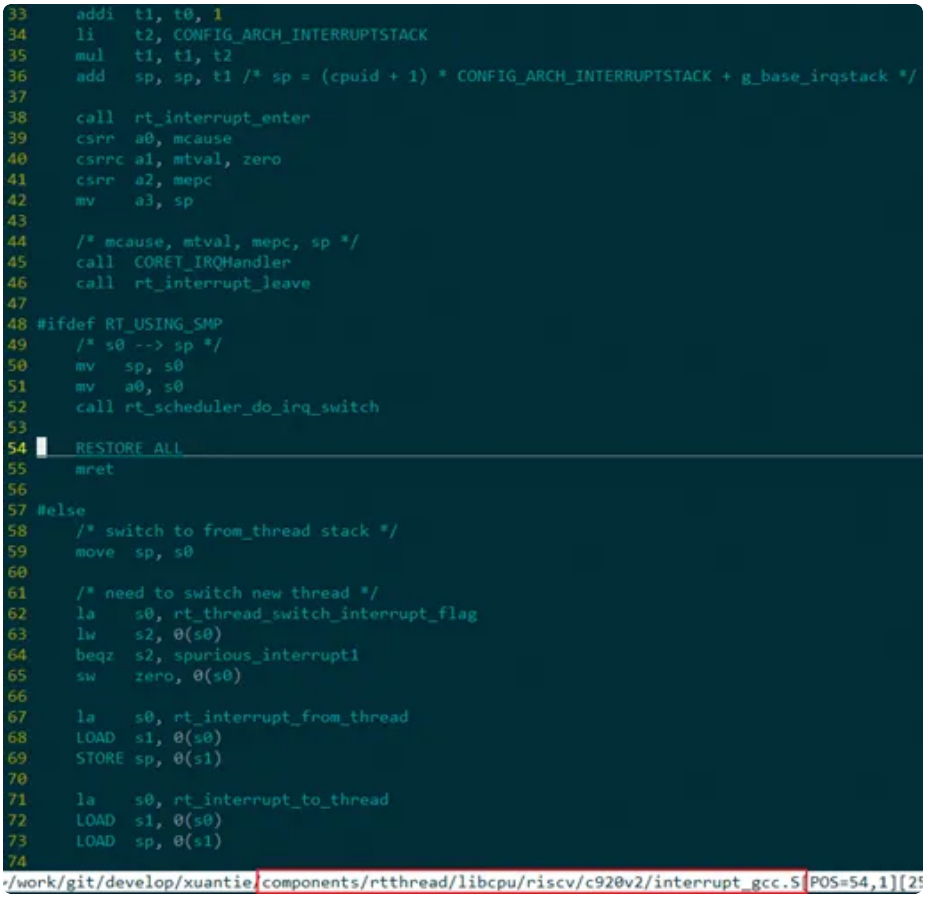
重要的移植接口请参考下表：

主要函数/变量	说明
rt_uint8_t *rt_hw_stack_init(void *tentry, void *parameter, rt_uint8_t *stack_addr, void *texit)	任务栈初始化，包含通用寄存器，浮点/向量/matrix寄存器(若需要)等。在创建任务时会被调用
void rt_hw_interrupt_enable(void)	使能全局中断
void rt_hw_interrupt_disable(void)	关闭全局中断
void rt_hw_context_switch(rt_ubase_t from, rt_ubase_t to)	从from线程切换到to线程，用于线程间的切换
void rt_hw_context_switch_interrupt(void *context, rt_ubase_t from, rt_ubase_t to, struct rt_thread *to_thread)	从from线程切换到to线程，用于在中断中切换任务
volatile rt_ubase_t rt_thread_switch_interrupt_flag = 0	表示需要在中断处理函数中进行任务切换的标志位

rt_uint32_t rt_interrupt_from_thread, rt_interrupt_to_thread;	在线程进行上下文切换时，用来保存from和to线程

以C920V2为例，





相关移植代码路径如下：

- 任务stackframe定义： components/rtthread/libcpu/riscv/c920v2/stack.h
- 任务栈初始化： components/rtthread/libcpu/riscv/c920v2/cpuport.c
- 定时器/软件/外部中断实现(覆盖chip组件中实现)：components/rtthread/libcpu/riscv/c920v2/interrupt_gcc.S
- IPI核间中断(用于SMP)相关实现： components/rtthread/libcpu/riscv/c920v2/clint.c
- 从核初始化(用于SMP)等： components/rtthread/libcpu/riscv/c920v2/cpuport_smp.c
- 板级配置头文件： boards/board_riscv_dummy/include/rtconfig.h

4.6.2.1. smp

当前除C906外，其他玄铁C/R9xx均支持smp。RTOS SDK中当前提供了两个smp示例(默认2 core)，分别为mcu_rtthread_smp和soc_smp_demo，用户可参考移植相关代码。

smp功能的使能主要依赖下面这两个配置：

```
# SMP CONFIG


CONFIG_SMP: 1
CONFIG_NR_CPUS: 2
```

5. 注意事项&使用tips

5.1. 注意事项

5.1.1. Linux下基于官方ZIP包示例编译失败

- 首先确认是否成功安装yoctools构建工具(保证pip和python版本均为python2/python3)
- 其次确认在解压根目录中是否执行过yoc init命令

5.1.2. 按键无法响应

使用minicom时，若无法响应键盘输入，需要将Serial port setup-->Hardware Flow Contorl选项改成No

5.1.3. CDS编译时出现找不到头文件的情况

出现这个问题的原因是mingw环境下对编译命令的长度有限制。用户可将SDK直接解压到某个盘的根目录下。错误参考如下：

```
ilding file: D:/support/csi-rtos/sdk-release/RTOS-SDK-C9xx-RV64-SMP/libs/libc/malloc.c
ilding file: D:/support/csi-rtos/sdk-release/RTOS-SDK-C9xx-RV64-SMP/libs/libc/minilibc_port.c
ilding file: D:/support/csi-rtos/sdk-release/RTOS-SDK-C9xx-RV64-SMP/libs/libc/printf.c
ilding file: D:/support/csi-rtos/sdk-release/RTOS-SDK-C9xx-RV64-SMP/csi_kernel/freertosv10.4.2/adapter/csi_freertos.c
ilding file: D:/support/csi-rtos/sdk-release/RTOS-SDK-C9xx-RV64-SMP/csi_kernel/freertosv10.4.2/FreeRTOS/Source/portable/MemMang/heap_4.c
ilding file: D:/support/csi-rtos/sdk-release/RTOS-SDK-C9xx-RV64-SMP/csi_kernel/freertosv10.4.2/FreeRTOS/Source/portable/GCC/riscv/port.c
ilding file: D:/support/csi-rtos/sdk-release/RTOS-SDK-C9xx-RV64-SMP/csi_kernel/freertosv10.4.2/FreeRTOS/Source/portable/GCC/riscv/portASM.S
ilding file: D:/support/csi-rtos/sdk-release/RTOS-SDK-C9xx-RV64-SMP/csi_kernel/freertosv10.4.2/FreeRTOS/Source/croutine.c
/support/csi-rtos/sdk-release/RTOS-SDK-C9xx-RV64-SMP/csi_kernel/freertosv10.4.2/FreeRTOS/Source/portable/GCC/riscv/portASM.S:70:10: fatal error: freert
 70 | #include "freertos_risc_v_chip_specific_extensions.h"
    |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
mpilation terminated.
ce: *** [csi_kernel/freertosv10.4.2/FreeRTOS/Source/portable/GCC/riscv/portASM.o] Error 1
ce: *** Waiting for unfinished jobs....
```

5.2. QEMU

5.2.1. 运行时缺少相关依赖包

基于ubuntu20.04运行qemu相关命令提示缺少相关依赖包时，请使用如下命令安装QEMU依赖的库：

```Bash
sudo apt-get install -y numactl libcurl4 libiscsi7 libaio1 libnfs13 librbd
1 libxkbcommon0 libfdt1 libpixman-1-0 libepoxy0 libpng16-16 libjpeg-turbo8
libsnappy1v5 liblzo2-2 libsdl2-2.0-0 libvdeplug2 libgtk-3-0 libcacard0 libb
rlapi0.7 libcapstone3 libspice-server1 libpmem1 libdaxctl1 libslirp0 libsdl
2-image-2.0-0 libvirglrenderer1 libusbredirhost1 libfuse3-3 libcurl3-gnutls
```

5.2.2. qemu执行后如何退出运行

先敲ctrl + a后，松开后紧接着敲x

5.2.3. 常用命令

以下是一些常用选项，更多的选项可以参考《QEMU Emulator User Documentation》。

```Plain Text
-help
显示帮助信息。
-version
显示版本信息。
-machine
选择模拟的开发板，可以输入-machine help 获取一个完整的开发板列表。
-cpu
选择CPU 类型（例如-cpu ck803），可以输入-cpu help 获取完整的CPU 列表。
-nographic
禁止所有的图形输出，模拟的串口将会重定向到命令行。
-gdb tcp::port
设置连接GDB 的端口，（例如-gdb tcp::23333, 将23333 作为GDB 的连接端口）
-S
在启动时冻结CPU ，（例如与-gdb 配合，通过GDB 控制继续执行）
```

6. 参考资料

相关文档和工具请从玄铁官方站点 https://www.xrvm.cn 下载

6.1. 文档

6.1.1. 《玄铁最小系统RTOS SDK快速上手手册》

资源地址：技术支持>资源下载>CPU处理器>玄铁9系列

6.2. 工具

6.2.1. 玄铁ELF工具链

资源地址：技术支持>资源下载>开发工具>编译工具：TAC>GCC-900系列

6.2.2. 玄铁QEMU模拟器

资源地址：技术支持>资源下载>开发工具>调试工具&模拟器>模拟器QEMU

6.2.3. 玄铁DebugServer调试工具

资源地址：技术支持>资源下载>开发工具>调试工具&模拟器>XuanTie Debug Server

6.2.4. CDS

资源地址：技术支持>资源下载>开发工具>集成开发环境：剑池CDS

6.2.5. CDK

资源地址：技术支持>资源下载>开发工具>集成开发环境：剑池CDK

6.3. yoctools和yaml规范

https://www.xrvm.cn/document?temp=yoctools&slug=yocbook

https://www.xrvm.cn/community/post/detail?spm=a2cl5.14300636.0.0.2e5fuleiuleiJX&id=3885147649007554560

https://www.xrvm.cn/community/post/detail?spm=a2cl5.14300636.0.0.2e5fuleiuleiJX&id=3885147854243241984

6.4. 更多组件

https://gitee.com/yocop