移植RT-Thread in Core-V-MCU

OPENHW

Core-v-mcu 简介

- core-v-mcu 文档资料

```
https://docs.openhwgroup.org/projects/core-v-mcu/index.html
```

- cv32e40p与RI5CY的关系

```
https://dingfen.github.io/risc-v/verilog/2020/07/16/RI5CY.html
```

- core-v-sdk

```
https://github.com/openhwgroup/core-v-sdk
```

- core-v-ide-cdt

```
https://github.com/openhwgroup/core-v-ide-cdt
```

- plct的qemu

```
https://github.com/plctlab/plct-qemu/tree/plct-corev-upstream-sync-dma
```

- cli_test

```
https://github.com/openhwgroup/core-v-mcu-cli-testc
```

- 项目目的

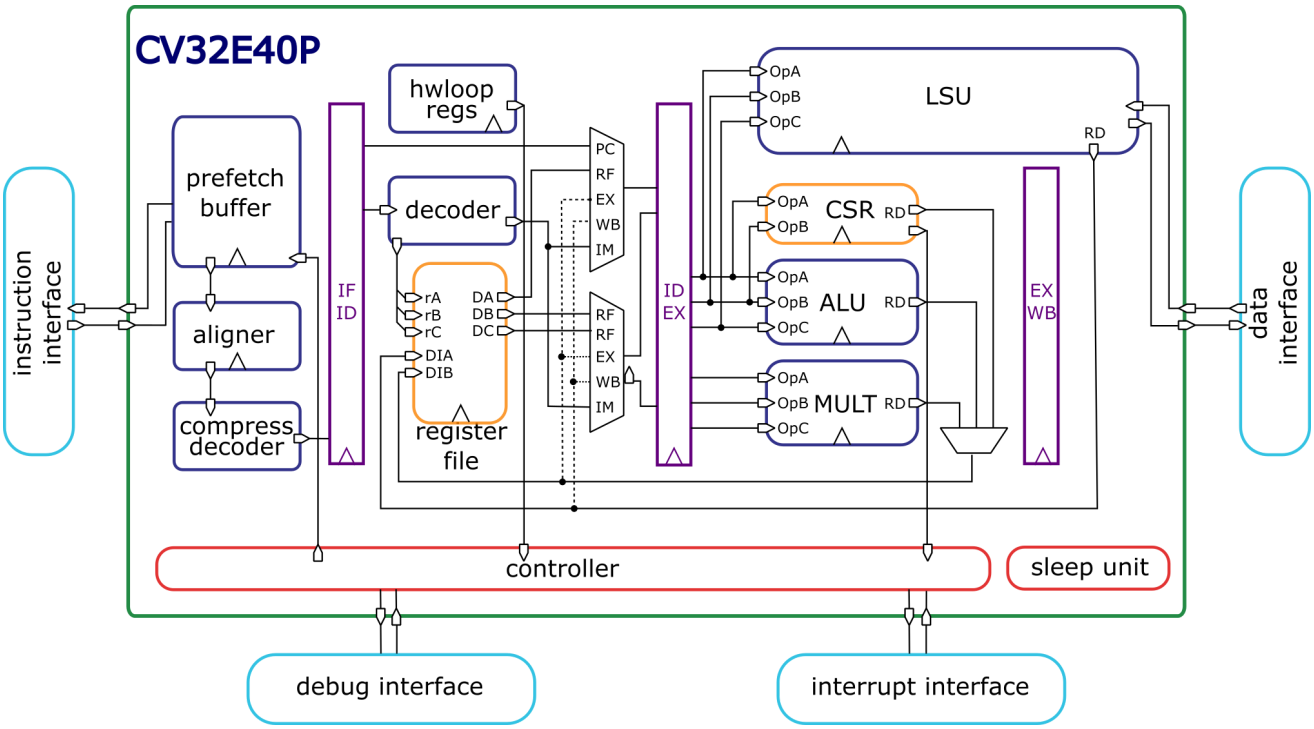
> *移植RT-Thread至core-v-mcu*

- 简介

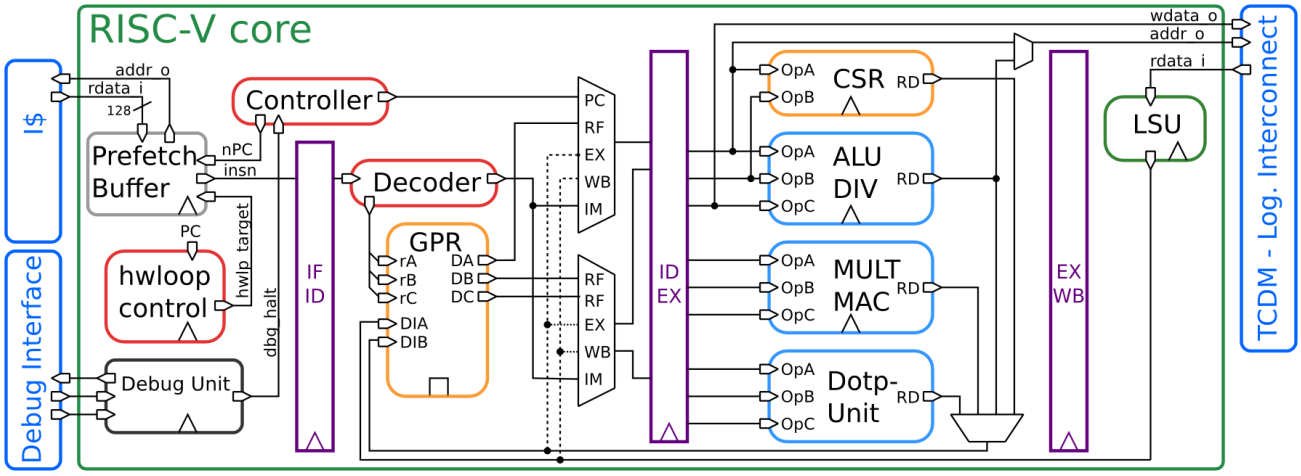
> - *core-v-mcu的内核为cv32e40p*
> - *cv32e40p继承自pulp开源的RI5CY内核 pulp是一个开源soc组织*
> - *openhw是一个内核开源组织 pulp加入openhw将RI5CY贡献给了openhw*
> - *core-v-mcu继承自PULPissimo PULPissimo是pulp维护的一个开源soc平台内核为RI5CY*

CV32E40P内核与RI5CY内核

- CV32E40P内核:

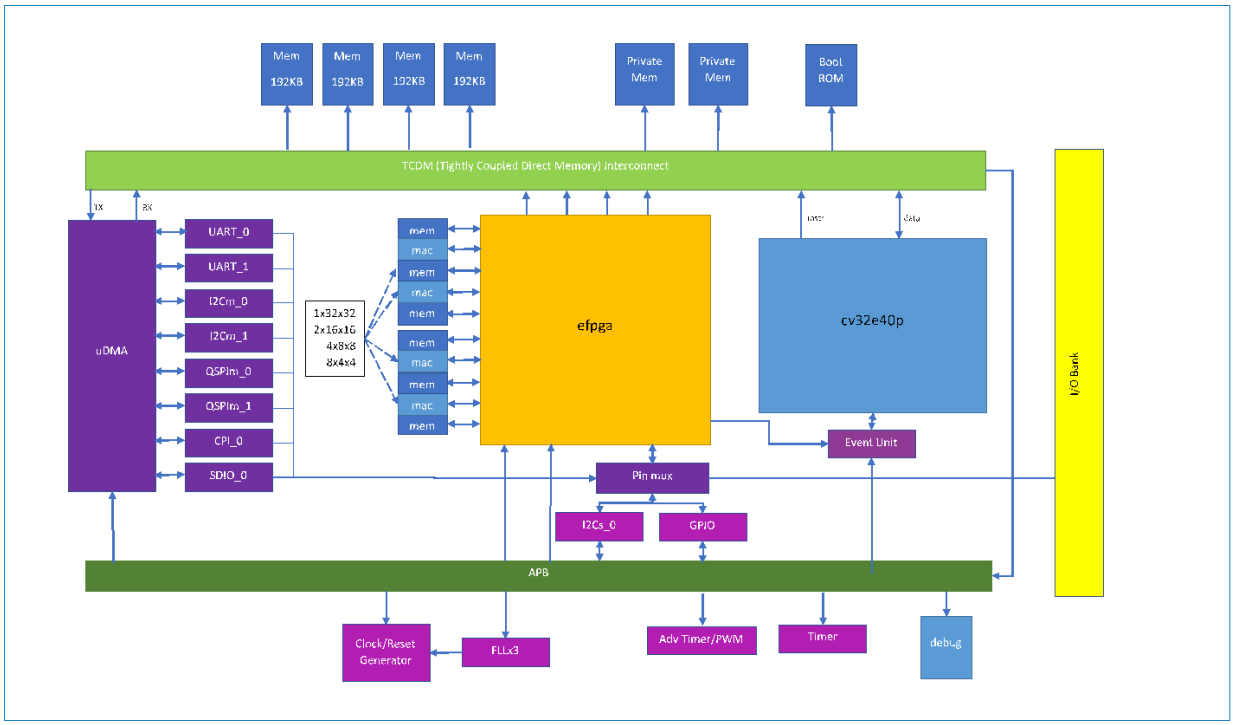


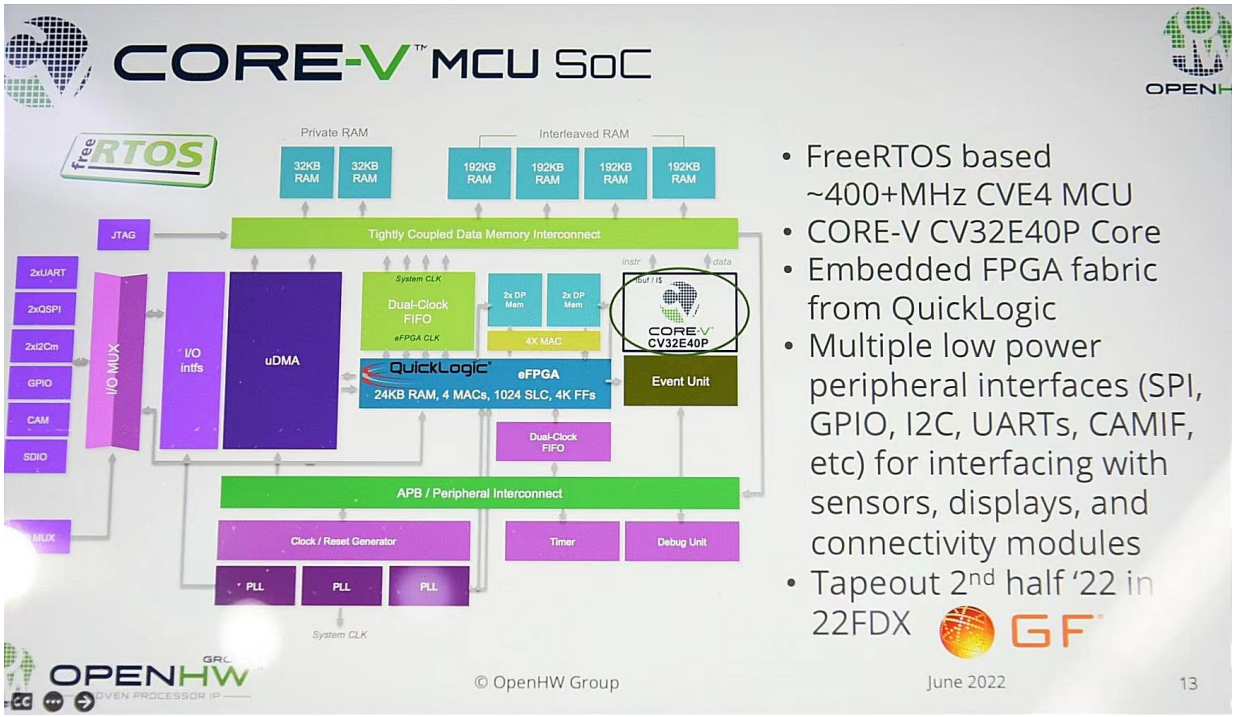
- RI5CY内核:



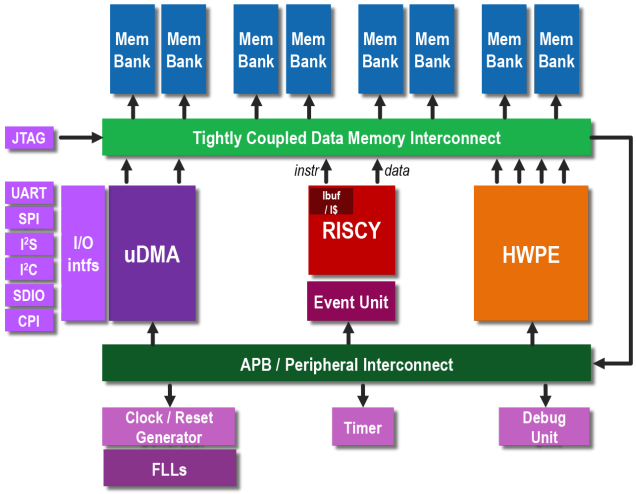
CORE-V-MCU与PULPissimo

- core-v-mcu





- PULLissimo



- OpenHW移植\core-v-mcu\CV32E40P_User_Manual-master 阅读笔记

> *CV32E40P是一个顺序（发射、执行）的四级流水线32位RISC-V处理器。CV32E40P的指令集包含了一部分的自定义扩展指令集，包括有：硬件循环（hardware loops）、地址自增的访存指令（post-increment load and store）以及额外的一系列ALU指令（算术指令扩展、乘累加MAC、向量操作等等）。core-v-mcu 文档资料*

Core-v-mcu 资源

core-v-mcu片上外设

core-v-mcu 的目的是展示OpenHW提供的 cv32e40p 完全验证的 RISC-V 内核。cv32e40p 内核连接到一组具有代表性的外设：

- 2路串口
- 2路I2C 主机
- 1路I2C 从机
- 2路QSPI 主机
- 1路相机
- 1路SDIO
- 4路PWM
- eFPGA

片上外设基地址与偏移

位于***core-v-mcu-pulp-mem-map.h***文件中

- 基地址

Description	Address Start	Address End
Boot ROM	0x1A000000	0x1A03FFFF
Peripheral Domain	0x1A100000	0x1A2FFFFF
eFPGA Domain	0x1A300000	0x1A3FFFFF
Memory Bank 0	0x1C000000	0x1C007FFF
Memory Bank 1	0x1C008000	0x1C00FFFF
Memory Bank Interleaved	0x1C010000	0x1C08FFFF

- 外设偏移

Description	Address Start	Address End
Frequency-locked loop	0x1A100000	0x1A100FFF
GPIOs	0x1A101000	0x1A101FFF
uDMA	0x1A102000	0x1A103FFF
SoC Controller	0x1A104000	0x1A104FFF
Advanced Timer	0x1A105000	0x1A105FFF
SoC Event Generator	0x1A106000	0x1A106FFF
I2CS	0x1A107000	0x1A107FFF
Timer	0x1A10B000	0x1A10BFFF
Stdout emulator	0x1A10F000	0x1A10FFFF
Debug	0x1A110000	0x1A11FFFF
eFPGA configuration	0x1A200000	0x1A2F0000
eFPGA HWCE	0x1A300000	0x1A3F0000

- 控制状态寄存器访问类型

Access Type	Description
RW	Read & Write
RO	Read Only
RC	Read & Clear after read
WO	Write Only
WC	Write Clears (value ignored; always writes a 0)
WS	Write Sets (value ignored; always writes a 1)
RW1S	Read & on Write bits with 1 get set, bits with 0 left unchanged
RW1C	Read & on Write bits with 1 get cleared, bits with 0 left unchanged
RW0C	Read & on Write bits with 0 get cleared, bits with 1 left unchanged

片上外设寄存器

- SOC-CTRL

> *基地址: SOC_CTRL_START_ADDR (0x1A104000)*
>
> *作用：配置连接在综合总线上的核的数量 、GPIO控制器上连接的IO的数量、UART的数量等，以及表明一些控制器的状态等*

- APB_EVENT_CNTRL

> *基地址:: SOC_EVENT_GEN_START_ADDR(SOC_EVENT_START_ADDR)*
>
> *作用： APB 外围设备收集所有呈现给 CPU 的事件作为 IRQ11（机器中断）。每个事件都可以通过 EVENT_MASKx 寄存器中的相应位单独屏蔽。当接收到启用事件（未屏蔽）时，它被放入事件 FIFO 中，并且 IRQ11 信号被提交给 CPU，然后 CPU 可以读取 EVENT FIFO 以确定哪个事件导致中断。每个事件都有一个深度为 4 的队列来收集事件，如果任何事件的队列溢出，则会将错误记录到相应的 EVENT_ERR 寄存器中，并将 IRQ31 提交给 CPU。*

- APB_TIMER_UINT

> *基地址: TIMER_START_ADDDR(0x1A10B000)*

- APB_GPIO

> *基地址: GPIO_START_ADDR(0x1A101000)、*

- APB_I2CS

> *基地址: I2CS_START_ADDR(0x1A107000)*

- eFPGA

  > *基地址: EFPGA_ASYNC_APB_START_ADD(EFPGA_ASYNC_APB_START_ADD)*

- UDMA_CTRL

  > *基地址: UDMA_CH_ADDR_CTRL(`UDMA_CH_ADDR_CTRL)*
  >
  > - *启用或禁用外设时钟*
  > - *重置外围控制器*
  > - *为事件处理机制设置比较值*
  > - ***core-v-mcu-config.h**中定义了UDMA 通道的起始 地址**UDMA_START_ADDR***

- UDMA_UART

  > *基地址:UDMA_CH_ADDR_UART(`UDMA_CH_ADDR_UART)*

- UDMA_I2CM

  > *基地址：UDMA_CH_ADDR_I2CM(UDMA_CH_ADDR_I2CM)*
  >
  > *I2C 控制器的动作是使用发送缓冲区中存在的一系列命令来控制的。因此，要使用I2C 控制器，软件必须在缓冲区中组装适当的命令序列，并使用UDMA 将缓冲区发送到 I2C 控制器。由于UDMA 处理数据缓冲区和中断，了解如何操作UDMA 控制器非常重要*

- UDMA_QSPI

  > *基地址:UDMA_CH_ADDR_QSPI(UDMA_CH_ADDR_QSPI)*
  >
  > *QSPI 控制器的动作是使用发送缓冲区中存在的一系列命令来控制的。因此，要使用 QSPI 控制器，软件必须在缓冲区中组装适当的命令序列，并使用UDMA 将缓冲区发送到QSPI 控制器。由于UDMA 处理数据缓冲区和中断，了解如何操作UDMA 控制器非常重要。*

- UDMA_SDIO

  > *基地址：UDMA_CH_ADDR_SDIO(`UDMA_CH_ADDR_SDIO)*

- UDMA_CAMERA

  > *基地址：UDMA_CH_ADDR_CAMERA(UDMA_CH_ADDR_CAMERA)*

串口映射到的管脚

IO_7		uart0_rx		apbio_0	fpgaio_0
IO_8		uart0_tx		apbio_1	FPGAIO_1
IO_9		uart1_tx		apbio_2	FPGAIO_2
IO_10		uart1_rx		apbio_3	FPGAIO_3

环境配置

- qemu 执行命令

```
./qemu-system-riscv32 -M core_v_mcu -bios none -kernel cli_test -
nographic -monitor none -serial stdio
```

- 配置与编译命令

```
进入
cli_test/app/ 目录
执行
source ../env/core-v-mcu.sh
make RISCV=xxx
注释:xxx为工具链的路径
```

- 安装工具链

> *工具链路径：OpenHW移植\可执行文件\toolchain*

- 修改工具链路径

> - *进入 cli_test/app/ 目录*
> - *查看makefile文件 找到default_flags.mk文件*

```
# indicate this repository's root folder
# set some project specific path variables
ifndef FREERTOS_PROJ_ROOT
$(error "FREERTOS_PROJ_ROOT is unset. Run source env/platform-you-want.sh \
        from the freertos project's root folder.")
endif


# good defaults for many environment variables
include $(FREERTOS_PROJ_ROOT)/default_flags.mk

# rtos and pulp sources, minimal
include $(FREERTOS_PROJ_ROOT)/metal_srcs.mkSs

# application name
PROG = cli_test
```

> - *打开default_flags.mk文件修改工具链的路径为使用的路径*

```
RISCV             ?= $(HOME)/gcc_riscv32/bin
RISCV_PREFIX      ?= $(RISCV)/riscv32-unknown-elf-
CC                = $(RISCV_PREFIX)gcc
OBJCOPY           = $(RISCV_PREFIX)objcopy
OBJDUMP           = $(RISCV_PREFIX)objdump
SIZE              = $(RISCV_PREFIX)size
```

- 配置python环境

```
wangshun@wangshun-virtual-machine:~/plct_cli/cli_test/cli_test/app$ python
Python 3.6.9 (default, Jun 29 2022, 11:45:57)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import elftools
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'elftools'
>>> exit()
wangshun@wangshun-virtual-machine:~/plct_cli/cli_test/cli_test/app$ pip3 install pyelftools
Collecting pyelftools
  Using cached https://files.pythonhosted.org/packages/04/7c/867630e6e6293793f838b31034aa1875e1c3bd8c1ec34a0929a2506f350c/pyelftools-0.29-py2.py3-none-any.whl
Installing collected packages: pyelftools
Successfully installed pyelftools-0.29
wangshun@wangshun-virtual-machine:~/plct_cli/cli_test/cli_test/app$ make
/home/wangshun/plct_cli/cli_test/cli_test/scripts/pulpstim -o cli_test.stim cli_test
/home/wangshun/gcc_riscv32/bin/riscv32-unknown-elf-objcopy -O ihex cli_test  cli_test.hex
/home/wangshun/gcc_riscv32/bin/riscv32-unknown-elf-objdump --source --all-headers --demangle --line-numbers --wide --prefix-addresses \
        cli_test > cli_test.lst
/home/wangshun/gcc_riscv32/bin/riscv32-unknown-elf-size --format=berkeley cli_test
   text    data     bss     dec     hex filename
 173112   68964   87024  329100   5058c cli_test
```

*先卸载**python2**,在安装**python3 elftools**工具在**python3**中的名称为**pyelftools**，修改方式如下,步骤**1**卸载**python2**软连接，安装**python3**软连接，步骤**2**执行第二个命令*

```
https://blog.csdn.net/u011304078/article/details/121430785
```

```
pip install pyelftools
```

寄存器详解

CV32E40P内核

- PULP指令支持

> ***FPU**:开启浮点支持 启用后支持单精度浮点*
>
> ***PULP_CLUSTER**： 启用pulp扩展集群支持*
>
> ***PULP_XPULP**:启用所有PULP扩展和自定义CSR*
>
> ***PULP_ZFINX**:启用PULP浮点支持而采用通用寄存器运算*
>
> ***WIF**:该指令可以使系统进入休眠*
>
> *HWloop： 硬件循环 开启**PULP_XPULP**后有效*

- 机器控制状态寄存器CSR

参考文章

RISC-V机器模式简介

RISC-V 机器模式下的寄存器与汇编指令

(0) 内核基础寄存器

Register Name	ABI Name	Description
x0	zero	Hard-Wired Zero
x1	ra	Return Address
x2	sp	Stack Pointer
x3	gp	Global Pointer
x4	tp	Thread Pointer
x5	t0	Temporary/Alternate Link Register
x6-7	t1-t2	Temporary Register
x8	s0/fp	Saved Register (Frame Pointer)
x9	s1	Saved Register
x10-11	a0-a1	Function Argument/Return Value Registers
x12-17	a2-a7	Function Argument Registers
x18-27	s2-s11	Saved Registers
x28-31	t3-t6	Temporary Registers

Table 1.1: RISC-V Base Integer Registers Of Size XLEN

(1) mastatus:机器状态寄存器

Machine Status Register(MSTATUS)寄存器详细描述了机器的状态，并帮助控制机器的状态。mstatus寄存器有几个位来控制机器的不同状态。

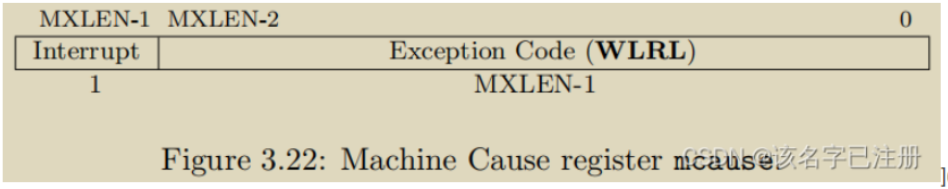


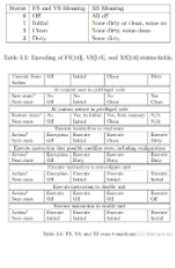
Figure 3.22: Machine Cause register mcause

](https://img-blog.csdnimg.cn/9f13268940924d19a647e48642e09384.png)

MSTATUS包含许多可以读取和更新的字段。通过修改这些字段，软件可以做一些事情，比如启用/禁用中断和更改虚拟内存模型等。
mstatus.MIE：Machine- Mode interrupt enable，机器模式全局中断使能位
mstatus.SIE： Supervisor-Mode interrupt enable，管理员模式全局中断使能位
x IE：=1则使能全局中断，=0则关闭全局中断，其只能控制小于或等于x模式下的中断，比如SIE=0，M模式下的中断不受其影响。
mstatus.MPIE：Machine- Mode previous interrupt enable，机器模式先前中断使能位
mstatus.SPIE：Supervisor-Mode previous interrupt enable，管理员模式先前中断使能位
x PIE：保存在trap之前interrupt-enable（x IE）位的值。
mstatus.MPP：Machine- Mode previous privilege，机器模式先前特权模式
mstatus.SPP：Supervisor- Mode previous privilege，管理员模式先前特权模式
x PP：保存trap之前的特权模式。xPP字段只能持有最多x的特权模式，因此MPP是2位宽，SPP是1位宽。
mstatus.MPRV：MPRV (Modify PRiVilege)位修改有效特权模式，即加载和存储执行时的特权级别。当MPRV=0时，使用当前特权模式的转换和保护机制，加载和存储行为正常。当MPRV=1时，加载和存储内存地址被转换和保护，并应用字节顺序，就好像当前特权模式被设置为MPP。指令地址转换和保护不受MPRV设置的影响。如果不支持U-mode，则MPRV为只读0。
当使用mret从trap中返回。会根据MPP的值来确定返回的新的特权模式，然后硬件改写mstatus中的MPP=0，MIE=MPIE，MPIE=进trap前的MIE，并设置PC=MEPC。

字段名称	bit	含义	功能	模式	RV
SIE	1	Global interrupt-enable bits	开关全局中断比特位, 可以使用csr指令set/cleared。 当高特权级的中断位被禁止时，低特权级的中断位无论是否开启，都将会被禁止	S-mode	RV32/64
MIE	3	Global interrupt-enable bits	开关全局中断比特位, 可以使用csr指令set/cleared	M	RV32/64
SPIE	5	the interrupt-enable bit active prior to the trap	在当前trap之前，中断的状态	S	RV32/64
MPIE	7			M	RV32/64

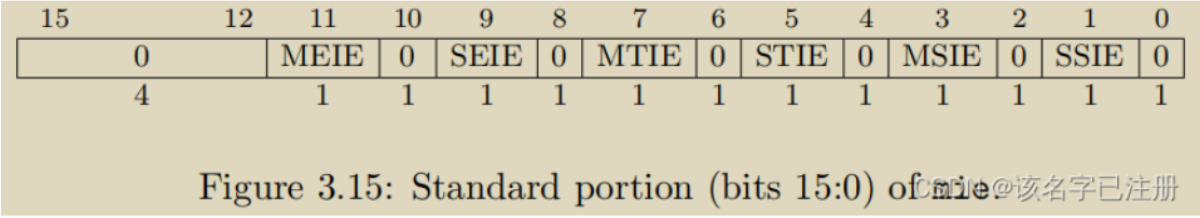
> *判断FS用于上下文切换时是否需要保存浮点寄存器*

FS	[14:13]		1.FS域描述浮点数单元状态，包括浮点数寄存器f0-f31和CSRs fcsr,frm和fflags		RV32/64	FS、VS的 WARL域 和XS只读域是用作降低context保存和恢复的成本，通过设置和跟踪当前浮点数单元和别的u模式扩展。 这些域可以快速的决定是否保存或恢复状态。如果保存或恢复是必须的，通常需要额外的指令和CSRs是需要和优化流程
VS	[10:9]		1.VS域描述向量扩展状态，包括向量寄存器v0-v31和CSRs vcsr, vxrm, vxsat, vstart, vl, vtype, 和vlenb		RV32/64	
XS	[16:15]		1.XS字段编码其他U模式扩展和相关状态的状态		RV32/64	

(2) mie:机器中断使能寄存器

MIE（Machine Interrupt Enable Register）

中断使能寄存器，区别于mstatus.MIE作为全局控制，MIE是局部控制。



15 12	11	10	9	8	7	6	5	4	3	2	1	0
0	MEIE	0	SEIE	0	MTIE	0	STIE	0	MSIE	0	SSIE	0
4	1	1	1	1	1	1	1	1	1	1	1	1

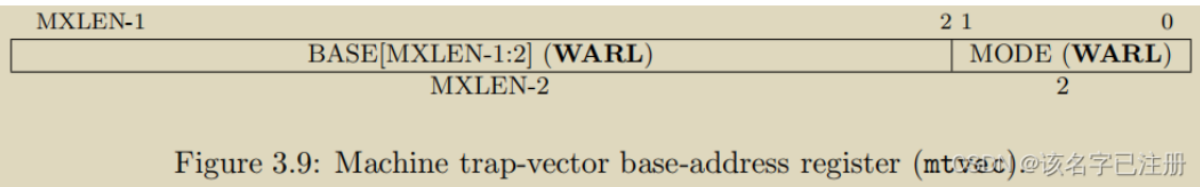
Figure 3.15: Standard portion (bits 15:0) of mie.

MEIE：M模式外部中断使能位
SEIE：S模式外部中断使能位
MTIE：M模式timer中断使能位
STIE：S模式外部中断使能位
MSIE：M模式软中断使能位
SSIE：S模式软中断使能位

(3) mtvec：机器Trap-Handler寄存器 （异常入口地址寄存器）

MTVEC（Machine Trap Vector Base Address register）

MTVEC用于存储Trap处理程序的地址。就是存储中断向量表的基地址。



MXLEN-1 2	1 0
BASE[MXLEN-1:2] (**WARL**)	MODE (**WARL**)
MXLEN-2	2

Figure 3.9: Machine trap-vector base-address register (mtvec).

Value	Name	Description
0	Direct	All exceptions set pc to BASE.
1	Vectored	Asynchronous interrupts set pc to BASE+4×cause.
≥2	—	*Reserved*

当MODE=Direct时，所有进入机器模式的trap都会导致pc被设置为BASE字段中的地址。当MODE= vector时，所有进入机器模式的同步异常都会导致pc被设置为BASE字段中的地址，而异步中断会导致pc被设置为BASE字段中的地址加上中断cause数的四倍。

(4) mscratch:

> *mscratch 寄存器用于机器模式下的程序临时保存某些数据。mscratch 寄存器可以提供一*
> *种快速的保存和恢复机制。譬如，在进入机器模式的异常处理程序后，将应用程序的某个通*
> *用寄存器的值临时存入 mscratch 寄存器中，然后在退出异常处理程序之前，将 mscratch 寄*
> *存器中的值读出恢复至通用寄存器。*

(5) mepc:

> *保存原PC，用于异常返回，可读写，软件可以更改*
> *中断时，mepc为下一条指令*
> *异常时，mepc为发生异常时当前的PC*

MEPC（Machine Exception Program Counter）

它保存导致trap的指令的地址。

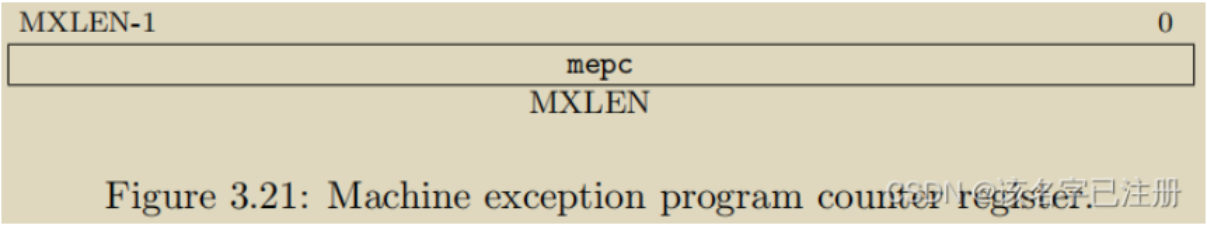


Figure 3.21: Machine exception program counter register.

(6) mcause: 异常产生原因寄存器

MCAUSE（Machine Cause Register）

Machine Cause Register寄存器是一个mxlen位的读写寄存器。当一个trap被带入m模式时，mcause被硬件写入一个代码，指示导致该trap的事件。如果trap是由中断引起的，则设置mcause寄存器中的中断bit位。

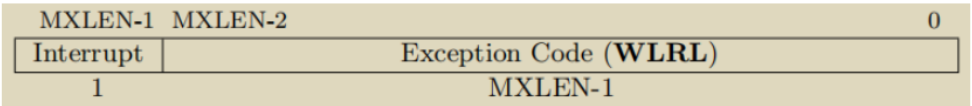


Figure 3.22: Machine Cause register mcause.

(7) mtval:

(8) mip:

MIP（Machine Interrupt Pending Register ）

中断挂起寄存器，包含关于挂起中断的信息。
我的理解：当正在处理一个中断，并且mie关掉中断时，同时设置了mip，此时产生了另一个中断则其会被pending，则MIP里对应的中断信息会被记录。

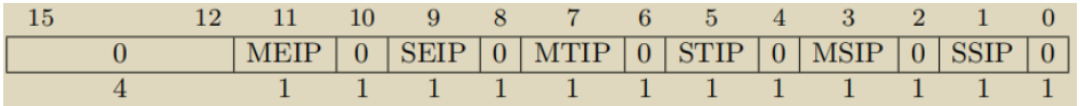


Figure 3.14: Standard portion (bits 15:0) of mip.

MEIP：M模式外部中断挂起位
SEIP：S模式外部中断挂起位
MTIP：M模式timer中断挂起位
STIP：S模式外部中断挂起位
MSIP：M模式软中断挂起位
SSIP：S模式软中断挂起位

(9) mcontext:

(10) scontext:

- 中断入口

  irq_i[11]、irq_i[7]、irq_i[3]分别对应机器外部中断（MEI）、机器定时器中断（MTI）、机器软件中断（MSI）

- CV32E40P会因为以下异常原因而触发异常

Exception Code	Description
2	Illegal instruction
3	Breakpoint
11	Environment call from M-Mode (ECALL)

- PC的注意事项

  对中断而言，mepc的值保存为下一条尚未执行的指令

  对异常而言，mepc的值被更新为当前发生异常的指令PC，这样有助于在异常服务程序中修正当前指令出现的错误；但是如果异常由ecall或ebreak造成，mepc的值会被更新为ecall或ebreak指令自己的PC，在指令返回时如果直接使用mepc保存的PC值将导致跳回ecall或ebreak导致死循环。需要在异常处理程序中用软件改变mepc=mepc+4或mepc=mepc+2

节拍定时器初始化入口

- OS心跳实现

```
void vPortSetupTimerInterrupt(void)
{
    extern int timer_irq_init(uint32_t ticks);

    /* No CLINT so use the PULP timer to generate the tick interrupt. */
    /* TODO: configKERNEL_INTERRUPT_PRIORITY - 1 ? */
    timer_irq_init(ARCHI_FPGA_FREQUENCY / configTICK_RATE_HZ);
    /* TODO: allow setting interrupt priority (to super high(?)) */
    //irq_enable(IRQ_FC_EVT_TIMER0_HI); // not needed as timer comes in
irq7
//  irq_enable (IRQ_FC_EVT_SW7);  // enable MTIME
}
```

cv32e40p未实现CLINT,使用片timer实现心跳节拍，CLINT

环境配置2

- 环境变量问题

  查看环境变量配置

```
wangshun@wangshun-virtual-machine:~/plct_cli/cli_test/cli_test/app$ source ../env/core-v-mcu.sh
wangshun@wangshun-virtual-machine:~/plct_cli/cli_test/cli_test/app$ source ../env/core-v-mcu.sh
wangshun@wangshun-virtual-machine:~/plct_cli/cli_test/cli_test/app$ cat ../env/core-v-mcu.sh
#!/usr/bin/env bash

# Copyright 2020 ETH Zurich
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# SPDX-License-Identifier: Apache-2.0
# Author: Robert Balas (balasr@iis.ee.ethz.ch)

# var that points to this project's root
ROOT=$(cd "$(dirname "${BASH_SOURCE[0]}")/.." && pwd)
export FREERTOS_PROJ_ROOT="$ROOT"
export FREERTOS_CONFIG_FAMILY="core-v-mcu"
```

查看环境变量

```
wangshun@wangshun-virtual-machine:~/plct_cli/cli_test/cli_test/app$ echo $ROOT
/home/wangshun/plct_cli/cli_test/cli_test
```

- 设置全局变量

*Makefile × | main.c | idle.c

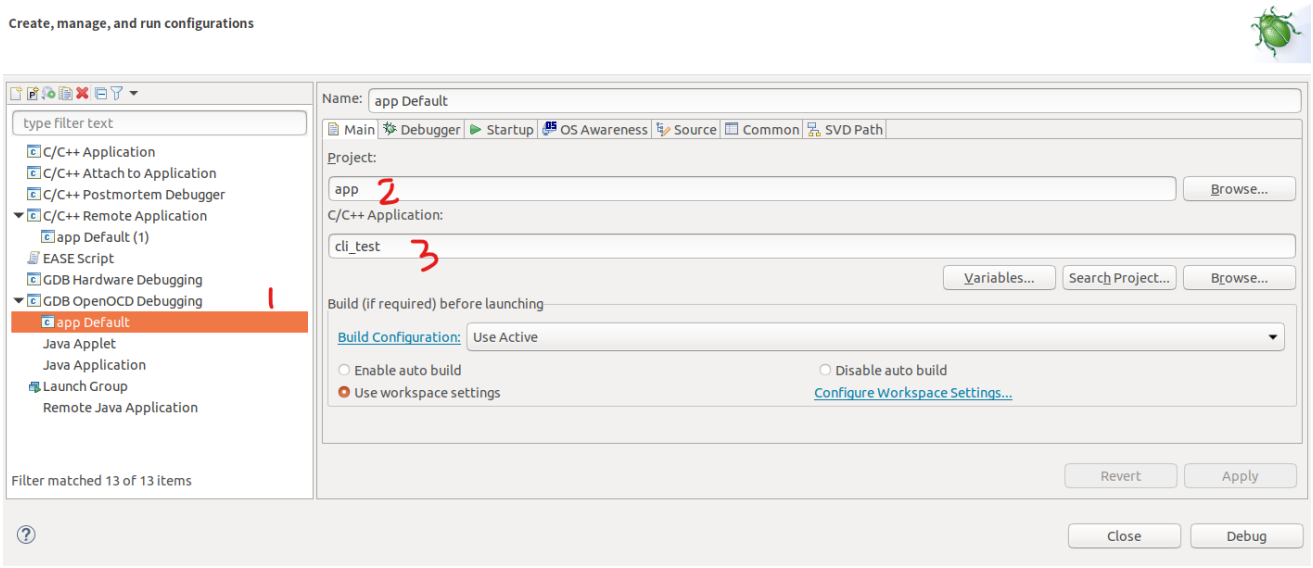
```
# LIBC       Link against libc (default yes)
# LTO        Enable link time optimization (default no)
# SANITIZE   Enable gcc sanitizer for debugging memory access problems (default no)
# STACKDBG   Enable stack debugging information and warnings.
#            By default 1 KiB but can be changed with MAXSTACKSIZE=your_value

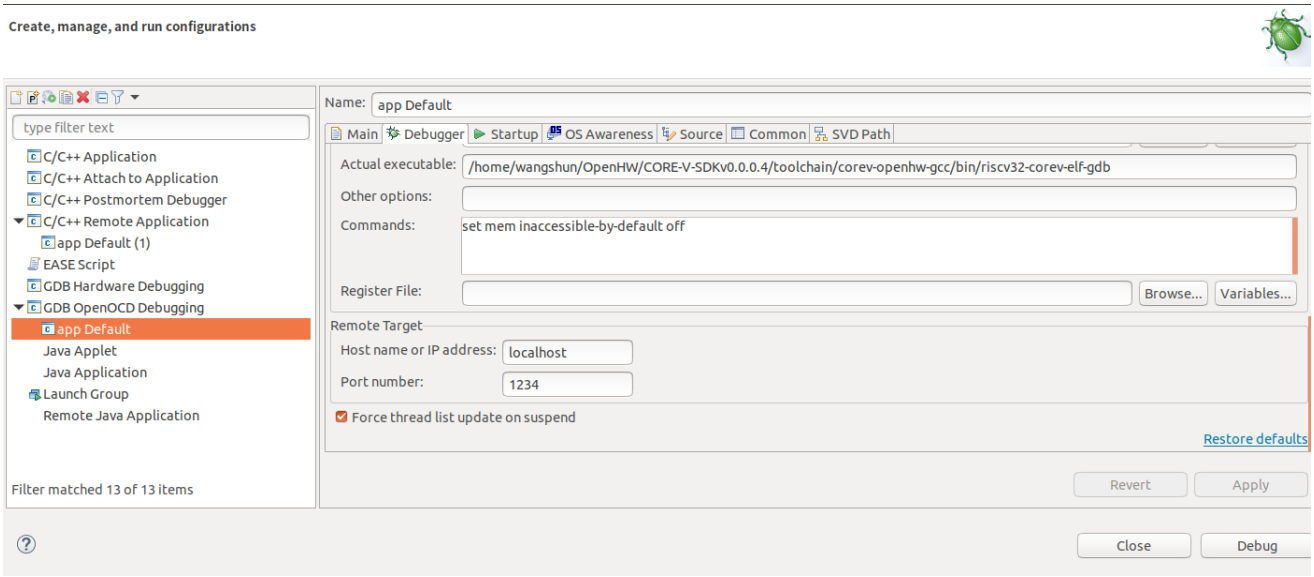

# indicate this repository's root folder
# set some project specific path variables

ROOT=/home/wangshun/plct_cli/cli_test/cli_test
FREERTOS_PROJ_ROOT=/home/wangshun/plct_cli/cli_test/cli_test
FREERTOS_CONFIG_FAMILY="core-v-mcu"

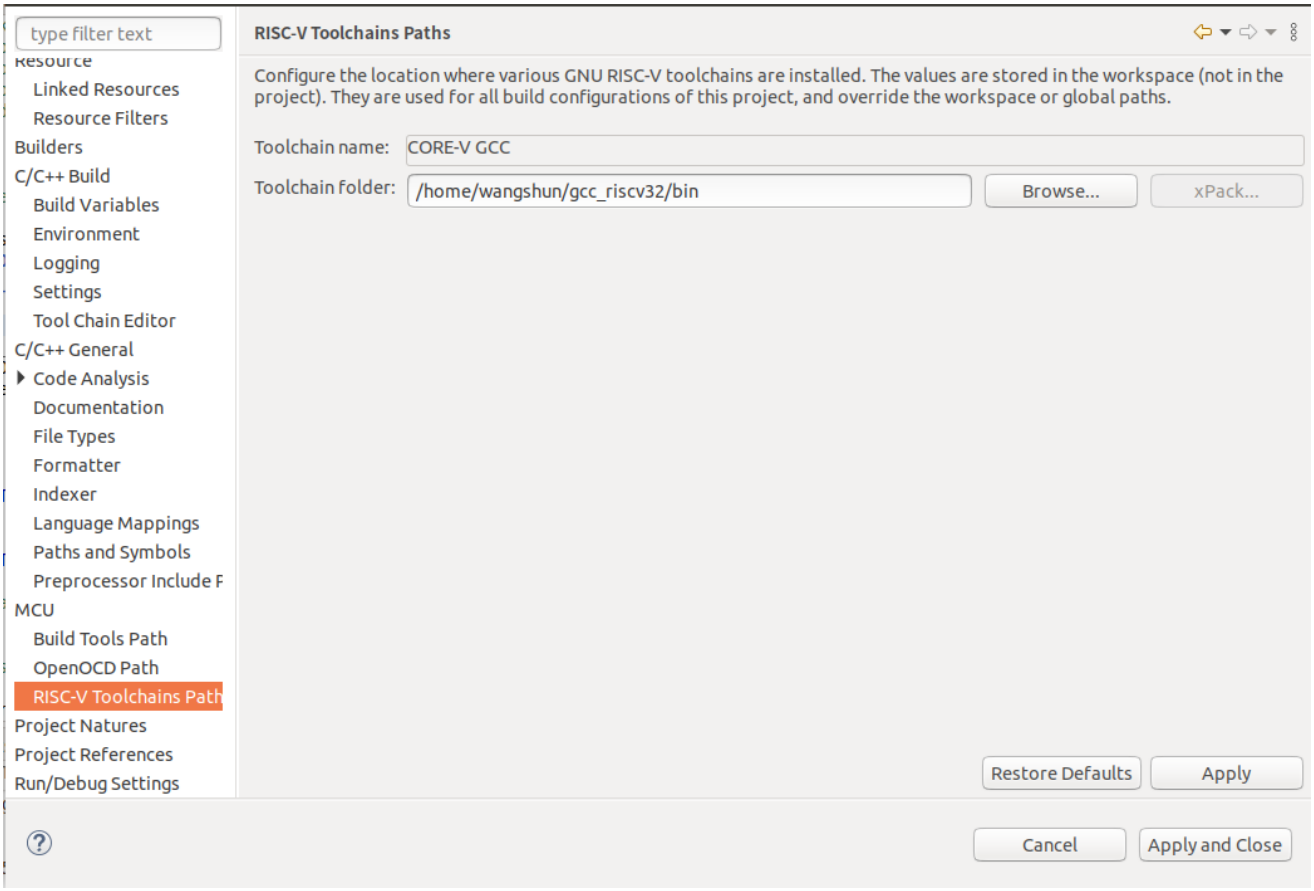

ifndef FREERTOS_PROJ_ROOT
$(error "FREERTOS_PROJ_ROOT is unset. Run source env/platform-you-want.sh \
    from the freertos project's root folder.")
endif
```

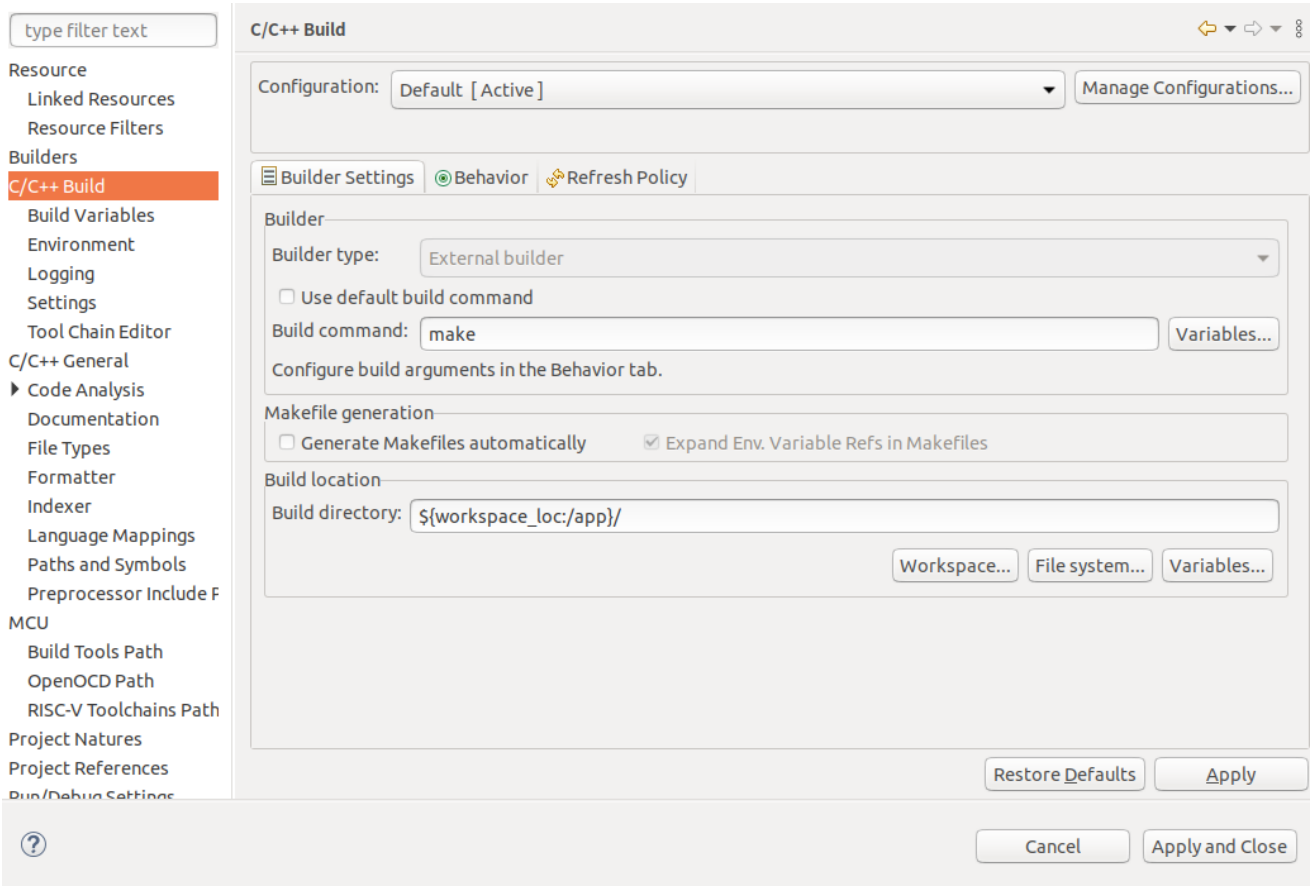
- 调试设置


Create, manage, and run configurations
type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
app Default (1)
EASE Script
GDB Hardware Debugging
GDB OpenOCD Debugging
app Default
Java Applet
Java Application
Launch Group
Remote Java Application
Filter matched 13 of 13 items
Name: app Default
Main
Debugger
Startup
OS Awareness
Source
Common
SVD Path
Project:
app
2
Browse...
C/C++ Application:
cli_test
3
Variables...
Search Project...
Browse...
Build (if required) before launching
Build Configuration:
Use Active
Enable auto build
Disable auto build
Use workspace settings
Configure Workspace Settings...
Revert
Apply
1
Close
Debug


Create, manage, and run configurations
type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
app Default (1)
EASE Script
GDB Hardware Debugging
GDB OpenOCD Debugging
app Default
Java Applet
Java Application
Launch Group
Remote Java Application
Filter matched 13 of 13 items
Name: app Default
Main
Debugger
Startup
OS Awareness
Source
Common
SVD Path
Actual executable:
/home/wangshun/OpenHW/CORE-V-SDKv0.0.0.4/toolchain/corev-openhw-gcc/bin/riscv32-corev-elf-gdb
Other options:
Commands:
set mem inaccessible-by-default off
Register File:
Browse...
Variables...
Remote Target
Host name or IP address:
localhost
Port number:
1234
Force thread list update on suspend
Restore defaults
Revert
Apply
Close
Debug

- 工具链设置





- qemu 运行指令

```
/home/wangshun/bin/qemu-riscv/bin/qemu-system-riscv32 -M core_v_mcu -
bios none -kernel cli_test -nographic -monitor none -serial stdio -S -s
```

RT-Thread移植

1.定时器节拍与中断向量表修改

- 在主函数中调用以下函数

```
    vPortSetupTimerInterrupt();//初始化定时器

    volatile uint32_t mtvec = 0;
    __asm volatile( "csrr %0, mtvec" : "=r"( mtvec ) );//声明仅有一张向量表
    __asm volatile( "csrs mie, %0" :: "r"(0x880) );//使能定时器中断与外部中断
    //note :task run always uless meet while(1)
```

- 修改中断向量表

未修改前

```
vector_table:
    j freertos_risc_v_trap_handler  // irq0
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler  // irq3
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler //ctxt_handler  // irq 7 mtime or timer
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler
    j h7// freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler // irq 11 Machine (event Fifo)
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler
    j freertos_risc_v_trap_handler // IRQ16
    j freertos_risc_v_trap_handler // IRQ17
    j freertos_risc_v_trap_handler // IRQ18
    j freertos_risc_v_trap_handler // IRQ19
    j freertos_risc_v_trap_handler // IRQ20
    j freertos_risc_v_trap_handler // IRQ21
    j freertos_risc_v_trap_handler // IRQ22
    j freertos_risc_v_trap_handler // IRQ23
    j freertos_risc_v_trap_handler // IRQ24
    j freertos_risc_v_trap_handler // IRQ25
    j freertos_risc_v_trap_handler // IRQ26
    j freertos_risc_v_trap_handler // IRQ27
    j freertos_risc_v_trap_handler // IRQ28
    j freertos_risc_v_trap_handler // IRQ29
    j freertos_risc_v_trap_handler // IRQ30
    j freertos_risc_v_trap_handler // IRQ30
```

修改后

```
vector_table:
    j IRQ_Handler // irq0
    j IRQ_Handler
    j IRQ_Handler
    j IRQ_Handler // irq3
    j IRQ_Handler
    j IRQ_Handler
    j IRQ_Handler
    j IRQ_Handler //ctxt_handler // irq 7 mtime or timer
    j IRQ_Handler
    j IRQ_Handler
    j h7// IRQ_Handler
    j IRQ_Handler // irq 11 Machine (event Fifo)
    j IRQ_Handler
    j IRQ_Handler
    j IRQ_Handler
    j IRQ_Handler
    j IRQ_Handler // IRQ16
    j IRQ_Handler // IRQ17
    j IRQ_Handler // IRQ18
    j IRQ_Handler // IRQ19
    j IRQ_Handler // IRQ20
    j IRQ_Handler // IRQ21
    j IRQ_Handler // IRQ22
    j IRQ_Handler // IRQ23
    j IRQ_Handler // IRQ24
    j IRQ_Handler // IRQ25
    j IRQ_Handler // IRQ26
    j IRQ_Handler // IRQ27
    j IRQ_Handler // IRQ28
    j IRQ_Handler // IRQ29
    j IRQ_Handler // IRQ30
j IRQ_Handler // IRQ30
```

- *修改中断栈地址*
  - *1的地址设置是有问题的 仅对比*
  - *2的地址正确*

1(不正确地址)修改前

interrupt_gcc.S

```
move  s0, sp

/* switch to interrupt stack */
la    sp, __stack

/* interrupt handle */
call  rt_interrupt_enter
csrr  a0, mcause
csrr  a1, mepc
mv    a2, sp
call  SystemIrqHandler
call  rt_interrupt_leave
```

(不正确地址)修改后

```
    STORE x27, 27 * REGBYTES(sp)
    STORE x28, 28 * REGBYTES(sp)
    STORE x29, 29 * REGBYTES(sp)
    STORE x30, 30 * REGBYTES(sp)
    STORE x31, 31 * REGBYTES(sp)

    move  s0, sp

    /* switch to interrupt stack */
    la    sp, xISRStackTop

    /* interrupt handle */
    call  rt_interrupt_enter
    csrr  a0, mcause
    csrr  a1, mepc
```

2(正确地址)修改后

正确地址的位置

```
ipt.c   vectors.S   port.c 2   cli_test.lst   main.c 3   core-v-mcu.ld
core-v-mcu > core-v-mcu.ld
    MAX(__data_begin + 0x800, __bss_end - 0x800));

.heap : ALIGN(16)
{
  __heap_start = .;
  /* . += __heap_size; */
  /* __heap_end = .; *//* Will be automatically filled by the ucHeap array */
  /* . = ALIGN(16); */
  KEEP(*(.heap))
  __heap_end = .;
  ASSERT((__heap_start + __heap_size < __heap_end), "Error (Linkerscript): Heap is too large");
} > L2


.stack : ALIGN(16)
{
  stack_start = .;
  __stack_bottom = .;
  . += __stack_size;
  __stack_top = .;
  __freertos_irq_stack_top = .; /* sytem stack */
  stack = .;
} > L2
```

修改后

```
main.c   core-v-mcu.c   vectors.S   components.c   irq.c   idle.c   interrupt_gcc.S
    STORE x25, 25 * REGBYTES(sp)
    STORE x26, 26 * REGBYTES(sp)
    STORE x27, 27 * REGBYTES(sp)
    STORE x28, 28 * REGBYTES(sp)
    STORE x29, 29 * REGBYTES(sp)
    STORE x30, 30 * REGBYTES(sp)
    STORE x31, 31 * REGBYTES(sp)

    move  s0, sp

    /* switch to interrupt stack */
    la    sp, __freertos_irq_stack_top

    /* interrupt handle */
    call  rt_interrupt_enter
    csrr  a0, mcause
    csrr  a1, mepc
    mv    a2, sp
    call  vSystemIrqHandler
    call  rt_interrupt_leave
```

2.动态内存移植

- 添加栈顶栈底地址

```
#if defined(RT_USING_USER_MAIN) && defined(RT_USING_HEAP)
    #define RT_HEAP_SIZE (2*1024)
    static rt_uint8_t rt_heap[RT_HEAP_SIZE];
    void *rt_heap_begin_get(void)
    {
        return rt_heap;
    }
    void *rt_heap_end_get(void)
    {
        return rt_heap + RT_HEAP_SIZE;
    }
#endif
```

> *Note:字节对齐，堆的大小应大于main线程的的栈*

- ```
  void rt_hw_board_init()
  {
      /* System Clock Update */
      system_init();

      vPortSetupTimerInterrupt();

      volatile uint32_t mtvec = 0;
      __asm volatile( "csrr %0, mtvec" : "=r"( mtvec ) );
      __asm volatile( "csrs mie, %0" :: "r"(0x880) );
      /* System Tick Configuration */
      //SysTick_Config(SystemCoreClock / RT_TICK_PER_SECOND);

      /* Call components board initial (use INIT_BOARD_EXPORT()) */
  #ifdef RT_USING_COMPONENTS_INIT
      rt_components_board_init();
  #endif

  #if defined(RT_USING_USER_MAIN) && defined(RT_USING_HEAP)
      rt_system_heap_init(rt_heap_begin_get(), rt_heap_end_get());
  #endif
  }
  ```

  这里执行堆的初始化；

  > *Note:初期运行错误 可以在创建main线程时采用静态的方式，若系统启动，则证明堆太小。*

3.自动初始化

1.自动初始化参考

2.自动初始化参考2
  ```

```
/home/wangshun/gcc_riscv32/bin/riscv32-unknown-elf-size --format=berkeley cli_test
   text    data     bss     dec     hex filename
  26448   66700   27204  120352   1d620 cli_test

15:28:45 Build Finished. 0 errors, 1357 warnings. (took 3s.490ms)

/home/wangshun/gcc_riscv32/bin/riscv32-unknown-elf-size --format=berkeley cli_test
   text    data     bss     dec     hex filename
  26428   66700   27204  120332   1d60c cli_test

15:29:46 Build Finished. 0 errors, 1357 warnings. (took 3s.690ms)
```

- 在链接脚本中添加如下代码

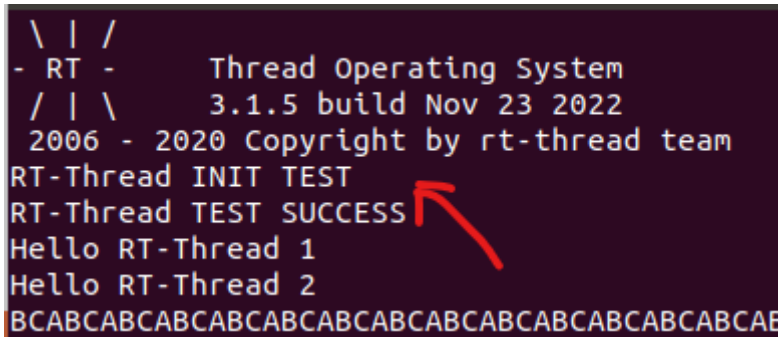
```
    .text __boot_address :
    {
      _stext = .;
      *(.text.start)
      *(.text)
      *(.text.*)
      _etext = .; /* man 3 end: first addr after text */
      *(.lit)
      *(.shdata)
      _endtext = .;
      . = ALIGN(4);

      /* section information for initial. */
      . = ALIGN(4);
      __rt_init_start = .;
      KEEP(*(SORT(.rti_fn*)))
      __rt_init_end = .;
      . = ALIGN(4);
    } > L2
```

- 添加测试代码

```
void led_init(void)
{
    rt_kprintf("RT-Thread INIT TEST \r\n");
    rt_kprintf("RT-Thread TEST SUCCESS \r\n");
}
INIT_APP_EXPORT(led_init);
```

- 测试结果



4.Finsh组件

shell移植参考1

shell移植参考2

- 在链接脚本中添加以下代码

```
.vectors MAX(0x1c000800, ALIGN(256)) : /* lets leak the first 2k free for now "half zero page" */
{
 __irq_vector_base = .;
 __vector_start = .;
 KEEP(*(.vectors))
} > L2

.text __boot_address :
{
 _stext = .;
 *(.text.start)
 *(.text)
 *(.text.*)
 _etext = .; /* man 3 end: first addr after text */
 *(.lit)
 *(.shdata)
 _endtext = .;
 . = ALIGN(4);

 /* section information for finsh shell */
 . = ALIGN(4);
 __fsymtab_start = .;
 KEEP(*(FSymTab))
 __fsymtab_end = .;
 . = ALIGN(4);
 __vsymtab_start = .;
 KEEP(*(VSymTab))
 __vsymtab_end = .;
 . = ALIGN(4);
```

- 轮询方式实现 rt_hw_console_getchar

```
char rt_hw_console_getchar(void)
{
    return udma_uart_getchar(0);
}
```

- Finsh测试

```
 \ | /
- RT -     Thread Operating System
 / | \     3.1.5 build Nov 23 2022
 2006 - 2020 Copyright by rt-thread team
RT-Thread INIT TEST
RT-Thread TEST SUCCESS
msh >
msh >
msh >
msh >
msh >
```

5.串口中断接收配置

- system_init函数中初始化串口0并绑定中断入口函数，中断入口函数绑定过程如下:

```
void system_init(void)

    /* TODO: enable uart */
    for (uint8_t id = 0; id != N_UART; id++) {
        udma_uart_open(id, 115200);
    }

uint16_t udma_uart_open (uint8_t uart_id, uint32_t xbaudrate) {
    UdmaUart_t*             puart;
    volatile UdmaCtrl_t*        pudma_ctrl = (UdmaCtrl_t*)UDMA_CH_ADDR_CTRL;

    /* See if already initialized */
    if (uart_semaphores_rx[uart_id] != NULL || uart_semaphores_tx[uart_id] != NULL) {
        return 1;
    }
    /* Enable reset and enable uart clock */
    pudma_ctrl->reg_rst |= (UDMA_CTRL_UART0_CLKEN << uart_id);
    pudma_ctrl->reg_rst &= ~(UDMA_CTRL_UART0_CLKEN << uart_id);
    pudma_ctrl->reg_cg |= (UDMA_CTRL_UART0_CLKEN << uart_id);

    /* Set semaphore */
    SemaphoreHandle_t shSemaphoreHandle;        // FreeRTOS.h has a define for xSemaphoreHandle, so can't use that
    shSemaphoreHandle = xSemaphoreCreateBinary();
    configASSERT(shSemaphoreHandle);
    xSemaphoreGive(shSemaphoreHandle);
    uart_semaphores_rx[uart_id] = shSemaphoreHandle;

    shSemaphoreHandle = xSemaphoreCreateBinary();
    configASSERT(shSemaphoreHandle);
    xSemaphoreGive(shSemaphoreHandle);
    uart_semaphores_tx[uart_id] = shSemaphoreHandle;

    /* Set handlers. */
    pi_fc_event_handler_set(SOC_EVENT_UART_RX(uart_id), uart_rx_isr, uart_semaphores_rx[uart_id]);
    pi_fc_event_handler_set(SOC_EVENT_UDMA_UART_TX(uart_id), NULL, uart_semaphores_tx[uart_id]);
    /* Enable SOC events propagation to FC. */
    hal_soc_eu_set_fc_mask(SOC_EVENT_UART_RX(uart_id));
    hal_soc_eu_set_fc_mask(SOC_EVENT_UDMA_UART_TX(uart_id));

    /* configure */
    puart = (UdmaUart_t*)(UDMA_CH_ADDR_UART + uart_id * UDMA_CH_SIZE);
    puart->uart_setup_b.div = (uint16_t)(5000000/xbaudrate);
    puart->uart_setup_b.bits = 3; // 8-bits
//  if (uart_id == 0) puart->uart_setup_b.rx_polling_en = 1;
//  if (uart_id == 1)
        puart->irq_en_b.rx_irq_en = 1;
    puart->uart_setup_b.en_tx = 1;
    puart->uart_setup_b.en_rx = 1;
    puart->uart_setup_b.rx_clean_fifo = 1;

    if (uart_id == 1) {
        u1rdptr = 0;
        u1wrptr = 0;
    }
    if (uart_id == 0) {
        u0rdptr = 0;
        u0wrptr = 0;
    }

void uart_rx_isr (void *id){
    if (id == 6) {
        while (*(int*)0x1a102130) {
            u1buffer[u1wrptr++] = puart1->data_b.rx_data & 0xff;
            u1wrptr &= 0x7f;
        }
    }
    if (id == 2) {
        while (puart0->valid) {
            u0buffer[u0wrptr++] = puart0->data_b.rx_data & 0xff;
            u0wrptr &= 0x7f;
        }
    }
}
```

- 串口中断收发测试

  修改串口中断函数

```
char n_data[]="\r\n";

void uart_rx_isr (void *id){
```

```
    if (id == 6) {

        while (*(int*)0x1a102130) {
            u1buffer[u1wrptr++] = puart1->data_b.rx_data & 0xff;
            u1wrptr &= 0x7f;
        }
    }

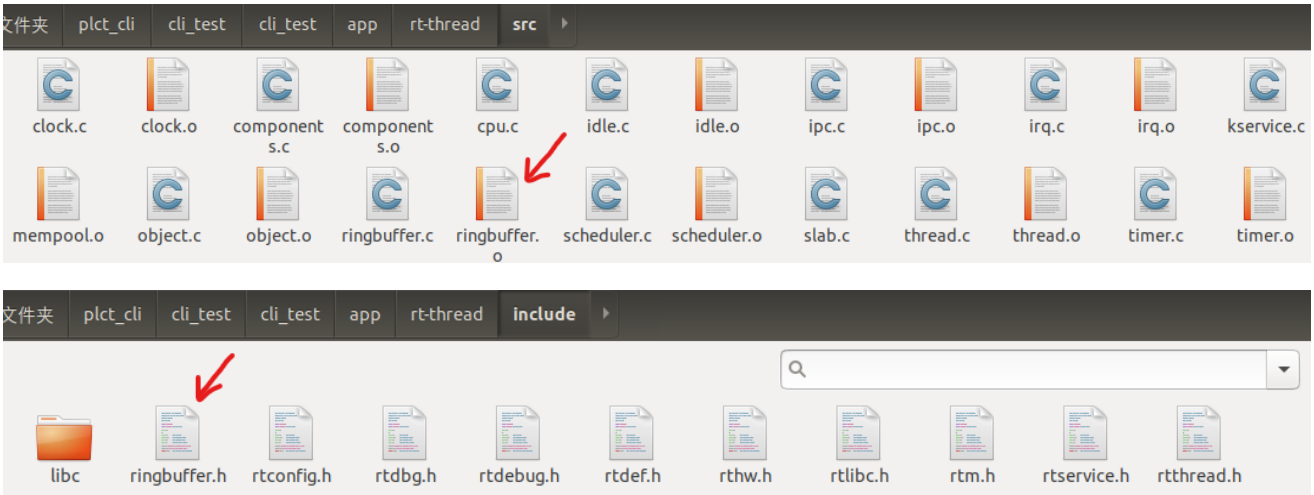
    if (id == 2) {
        while (puart0->valid) {
            u0buffer[u0wrptr++] = puart0->data_b.rx_data & 0xff;
            u0wrptr &= 0x7f;
            outdata(0,sizeof(u0buffer),u0buffer);
            outdata(0,sizeof(n_data),n_data);
            u0wrptr=0;
        }
    }
}
```

- 测试结果

```
wangshun@wangshun-virtual-machine:~/plct_cli/cli_test/cli_test/app$ /home/wangshun/bin/qemu-riscv/
bin/qemu-system-riscv32 -M core_v_mcu -bios none -kernel cli_test -nographic -monitor none -serial
 stdio
1
2
1
2
```

ringbuffer移植

- 1.添加ringbuffer文件至工程，修改makefile



- 修改ringbuffer.c

> *移除#include <device.h>*
>
> *添加#include <ringbuffer.h>*

- 修改中断入口函数

```
// ringbuffer

#define UART_RX_BUFFER_LEN 16

rt_uint8_t uart_rxbuffer[UART_RX_BUFFER_LEN]={0};
struct rt_ringbuffer uart_rxTCB;
struct rt_semaphore  shell_rx_semaphore;


char n_data[]="\r\n";
void uart_rx_isr (void *id){

    rt_interrupt_enter();
    if (id == 6) {
        while (*(int*)0x1a102130) {
            u1buffer[u1wrptr++] = puart1->data_b.rx_data & 0xff;
            u1wrptr &= 0x7f;
        }
    }

    if (id == 2) {
        while (puart0->valid) {
            rt_ringbuffer_putchar(&uart_rxTCB,puart0->data_b.rx_data
& 0xff);
        }
        rt_sem_release(&shell_rx_semaphore);
    }
    rt_interrupt_leave();
}
```

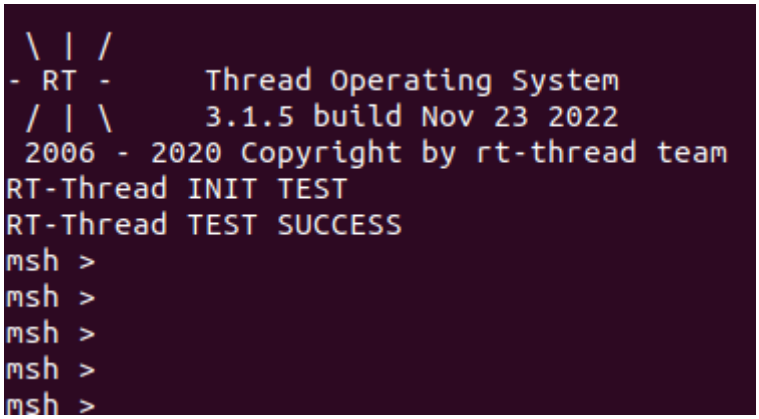
- 修改rt_hw_console_getchar函数

```
char rt_hw_console_getchar(void)
{
    char ch=0;
    while(rt_ringbuffer_getchar(&uart_rxTCB,(rt_uint8_t*)&ch)!=0)
    {
        rt_sem_take(&shell_rx_semaphore,RT_WAITING_FOREVER);
    }
    return ch;
    //return udma_uart_getchar(0);
}
```

- 在rt_hw_board_init初始化ringbuffer，创建信号量，添加如下代码

```
    rt_ringbuffer_init(&uart_rxTCB,uart_rxbuffer,16);
    rt_sem_init(&(shell_rx_semaphore),"shell_rx",0,0);
```

- 测试


\ | /
- RT -   Thread Operating System
/ | \   3.1.5 build Nov 23 2022
2006 - 2020 Copyright by rt-thread team
RT-Thread INIT TEST
RT-Thread TEST SUCCESS
msh >
msh >
msh >
msh >
msh >