2022.11.15

- 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

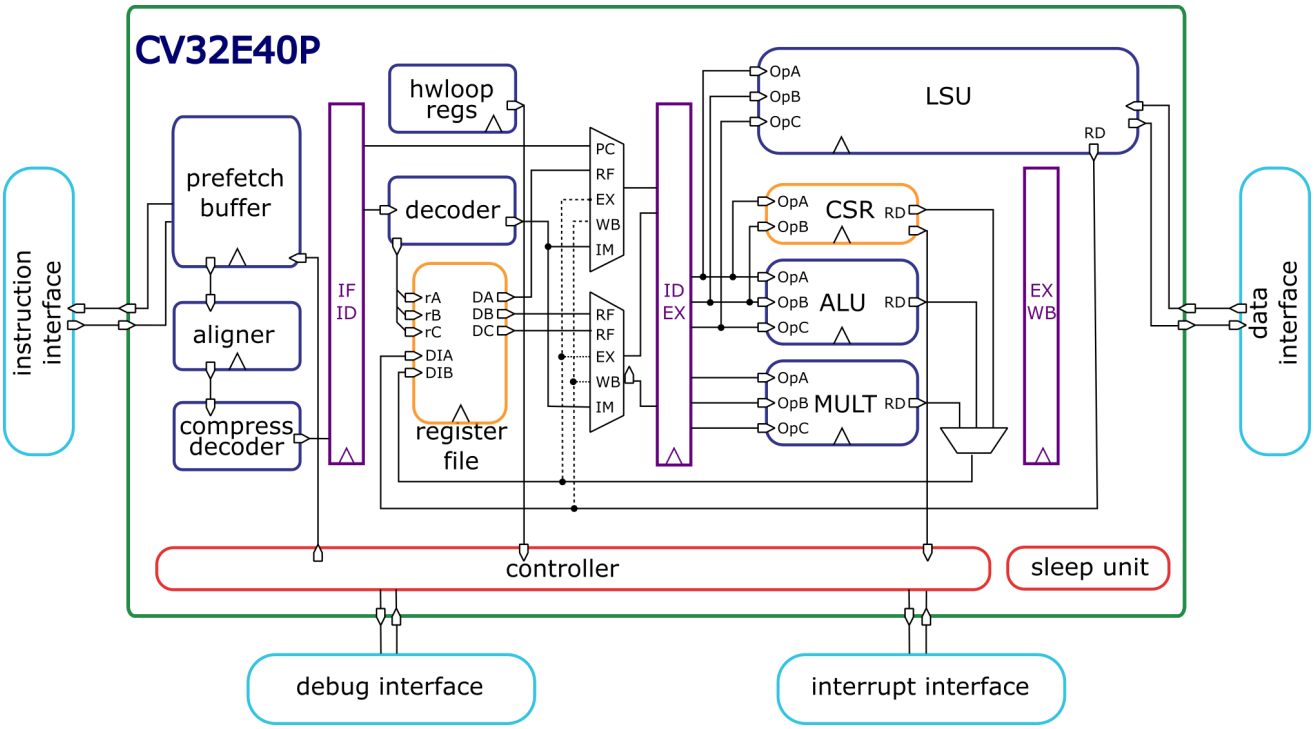
> *移植RT-Thread至core-v-mcu*

- 简介

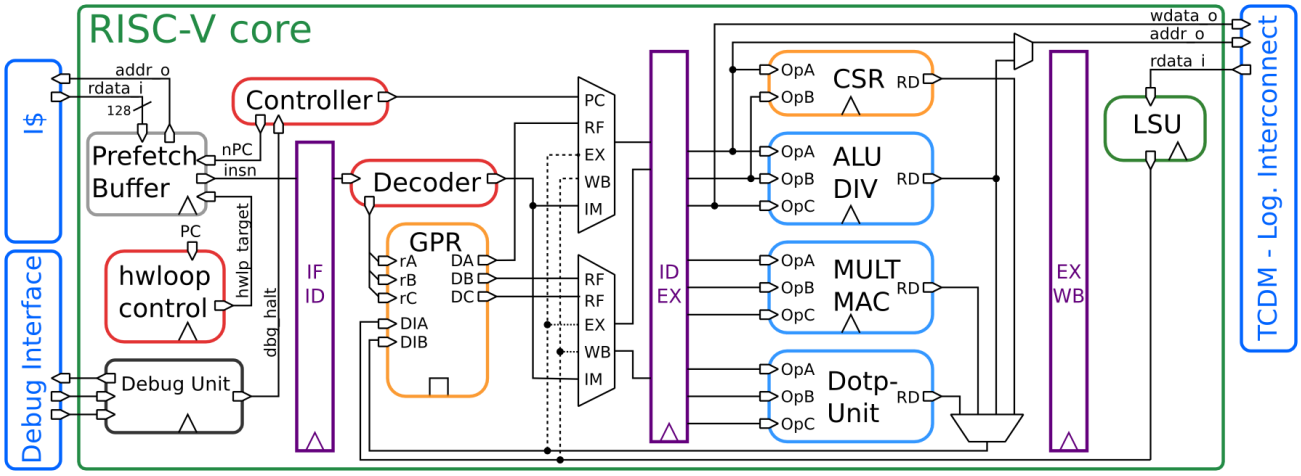
> - *core-v-mcu的内核为cv32e40p*
> - *cv32e40p继承自pulp开源的RI5CY内核 pulp是一个开源soc组织*
> - *openhw是一个内核开源组织 pulp加入openhw将RI5CY贡献给了openhw*
> - *core-v-mcu继承自PULPissimo PULPissimo是pulp维护的一个开源soc平台内核为RI5CY*

CV32E40P内核与RI5CY内核

- CV32E40P内核:

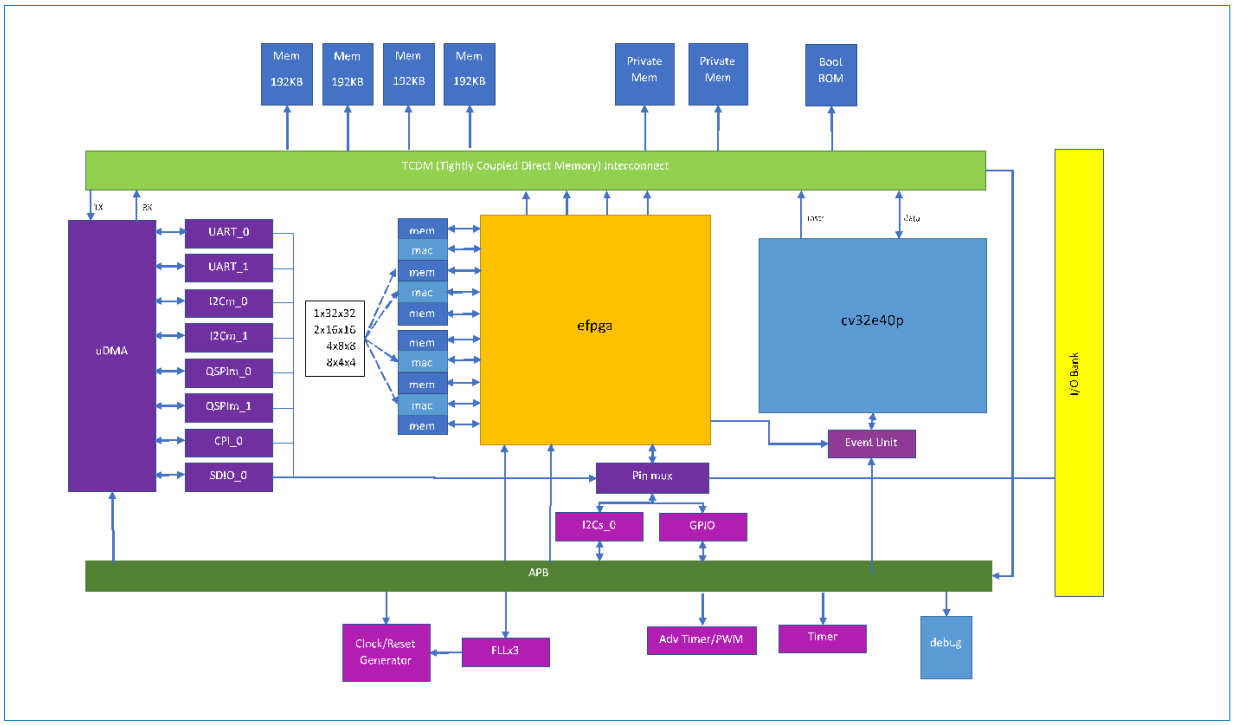


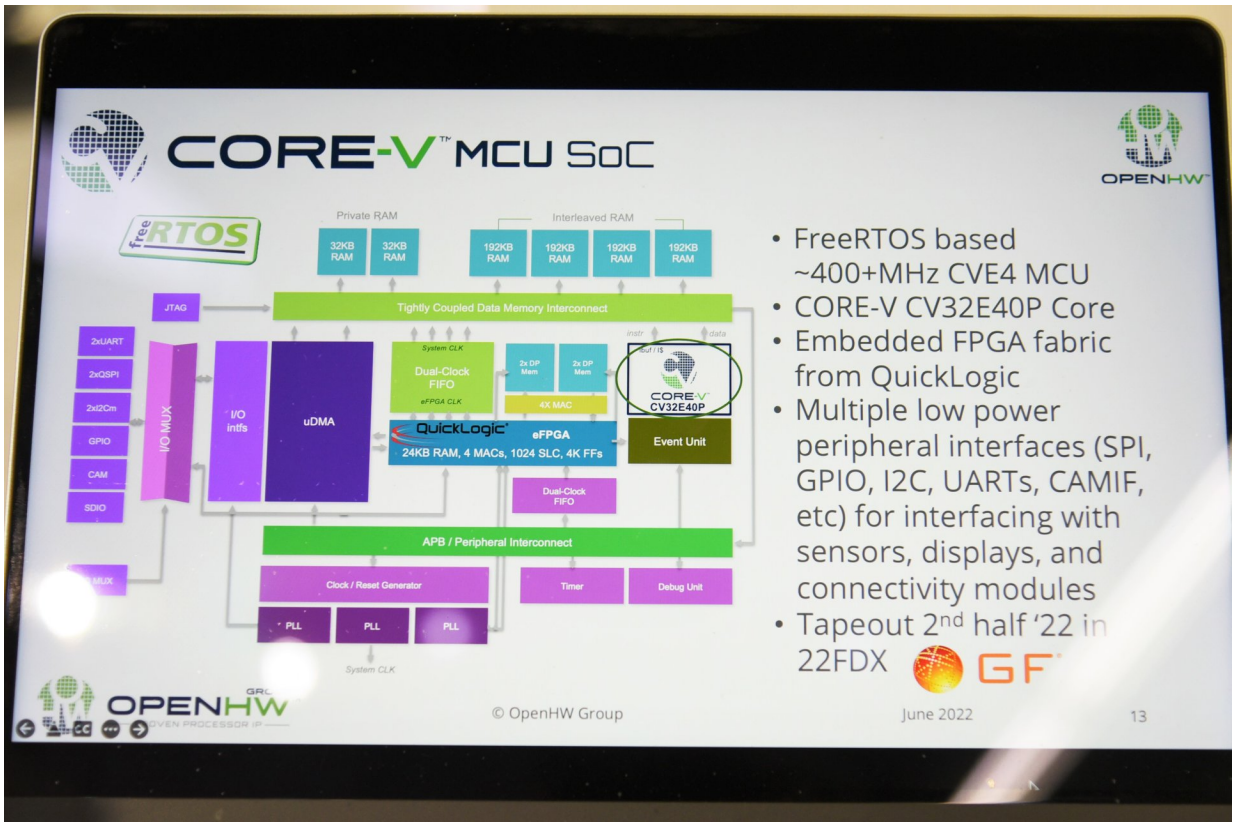
- RI5CY内核:



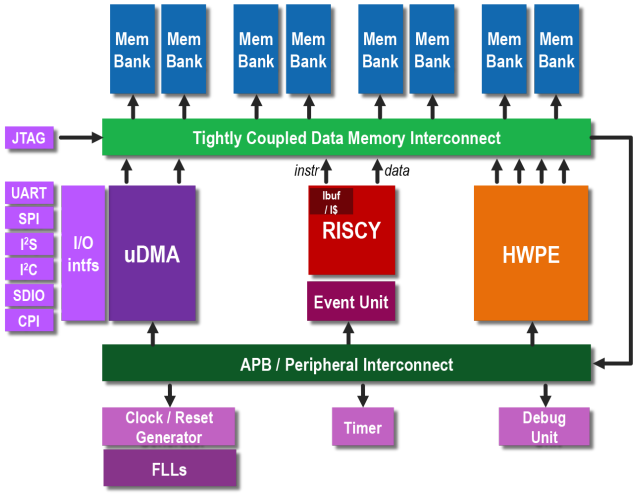
CORE-V-MCU与PULPissimo

- 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 文档资料笔记

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

2022.11.16

- 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
```

-