

零、知识点：

1. ECC: Error Checking and Correction

1. 保护SRAM，可以对1位错误纠正，2位错误上报系统；
2. 蜂鸟中对ITCM、DTCM中SRAM进行保护；

1、ram

1. sirv_sim_ram: 仿真SRAM，将x不定态->0

- 参数：
 - FORCE_X2ZERO: 有效->输出不定态变为0;
 - DP:
 - DW: din width;
 - AW: addr width;
 - MW: mask width;
- 信号接口：
 - wem: write enable mask, 默认4bits对应写入数据4byte, 写入掩码每一位对应;
 - cs: 有效才可以进行读写;
 - we: write enable; 有效->write, 无效->read;

2. sirv_gnrl_ram: ram顶层

- BUG: sd、ds、ls, 无用信号;
- sirv_gnrl_dfflr: 通过always块编写寄存器逻辑;
- | | |
|------------------|---|
| sirv_gnrl_dfflrs | //带有 load-enable 使能, 带有异步 reset, 复位默认值为 1 的寄存器 |
| sirv_gnrl_dfflr | //带有 load-enable 使能, 带有异步 reset, 复位默认值为 0 的寄存器 |
| sirv_gnrl_dffl | //带有 load-enable 使能, 不带有 reset 的寄存器 |
| sirv_gnrl_dffrs | //不带有 load-enable 使能, 带有异步 reset, 复位默认值为 1 的寄存器 |
| sirv_gnrl_dffr | //不带有 load-enable 使能, 带有异步 reset, 复位默认值为 0 的寄存器 |
| sirv_gnrl_ltch | //Latch 锁存器模块 |
- sirv_gnrl_icb_n2w: 位宽转换模块 将lsu32位换为64位;
- sirv_gnrl_icb_arbt: ICB总线仲裁模块;
- sirv_gnrl_fifo: 通过配置DP大小, 从而得到不同深度的fifo;
 - 0->单口ram;
 - 1: ready信号与下一级ready信号相关, 需要使用CUT_ready控制->1可以切断反压信号, 两个周期传递一次数据;
 - 大于1: fifo, CUT_READY无实际意义;
 - 在蜂鸟中, DP只有1或者2, 也就是说要么是控制ready信号, 反压; 要么是一个深度为2的fifo;

3. buf

(1). sirv_gnrl_pipe_stage: DP=0, o_dat=i_dat;DP=1

- 相当于将输入信号打一拍；

3. sirv_sram_icb_ctrl:

- 定义: *The icb_ecc_ctrl module control the ICB access requests to SRAM ;*
 - 内部定义一个bypbuf (含有fifo的bypass buffer) , 悬空一级流水线, 以减少反压ready信号-- (通过将输入、输出信号拼接成一个信号传输) ;
- 参数:
 - sram_ctrl_active: 高电平sram工作;
 - tcm_cgstop: 来自csr mcgstop (0xBFE) 控制, 蜂鸟自定义, 主要用于禁用在debug 中 ITCM中的SRAM门控时钟;

一、架构:

- e203
 - core
 - debug
 - fab
 - general
 - mems
 - perips: 存放外设的RTL代码;
 - soc
 - subsys: 存放子系统的RTL代码;

```

|----general      // 存放一些公用的通用 RTL 代码
|----core         // 存放 e203 Core 的 RTL 代码
|----fab          // 存放总线结构 (bus fabric) 的 RTL 代码
    |---- sirv_icb1to4_bus.v // 将 1 组 ICB 总线转换成 4 路 ICB 总线
    |---- sirv_icb1to8_bus.v // 将 1 组 ICB 总线转换成 8 路 ICB 总线
    |---- sirv_icb1to16_bus.v // 将 1 组 ICB 总线转换成 16 路 ICB 总线
|----subsys       // 存放完整子系统顶层的 RTL 代码
    |---- e203_subsys_top.v     // 子系统的顶层
    |---- e203_subsys_main.v   // 子系统的主体部分 (可关电) 顶层
    |---- e203_subsys_plic.v  // PLIC 顶层
    |---- e203_subsys_clint.v // CLINT 顶层
    |---- e203_subsys_mem.v   // 子系统的存储部分顶层
    |---- e203_subsys_perip.v // 子系统的外设部分顶层
|----mems         // 存放存储器模块的 RTL 代码
|----perip        // 存放外设 (peripherals) 模块的 RTL 代码
    |---- sirv_aon*.v           // 常开域部分模块
    |---- sirv_clint*.v         // CLINT 的模块
    |---- sirv_flash_qspi*.v   // Flash 专用的 QSPI 模块
    |---- sirv_gpio*.v          // GPIO 的模块
    |---- sirv_plic*.v          // PLIC 的模块
    |---- sirv_pmu*.v           // PMU 的模块
    |---- sirv_pwm16*.v          // 16bits 精度的 PWM 模块
    |---- sirv_pwm8*.v           // 8bits 精度的 PWM 模块
    |---- sirv_qspi_1cs*.v       // 1 个 CS 选通的 QSPI 模块
    |---- sirv_qspi_4cs*.v       // 4 个 CS 选通的 QSPI 模块
    |---- sirv_qspi*.v           // 其他 QSPI 子模块
    |---- sirv_rtc*.v             // RTC 模块
    |---- sirv_uart*.v            // UART 模块
    |---- sirv_wdog*.v            // WatchDog 模块
|----debug        // 存放调试相关模块的 RTL 代码
|----soc          // 存放 soc 顶层的 RTL 代码
    |----e203_soc_top.v         // SoC 顶层

```

1、整体特性：

1. 2级流水线处理器核； RV32IEMAC；
2. 机器模式；
 1. CLINT：计时器中断、软件中断；
 2. PLIC：外部中断， Platform Level Interrupt Controller； 用于多个外部中断源的优先级仲裁和派发；
 1. 是一个存储器地址映射 (Memory Address Mapped) 的模块，挂载在处理器核为其实现的专用总线接口上；
 3. 地址非对齐 (Address Misalign)：与 Rocket Core 采用软件支持， AGU 通过对生成出的访存地址进行判断，如果其地址非对齐，则产生异常标志；

4. 自定义CSR: mcounterstop

域	位	描述
CYCLE	0	<p>此位控制 mcycle 和 mcycleh 对应的计数器：</p> <ul style="list-style-type: none"> 如果此位为 1，则将计数器停止计数 如果此位为 0，则计数器正常工作 <p>此位上电复位默认值为 0</p>
TIMER	1	<p>此位控制 mtime 对应的计数器：</p> <ul style="list-style-type: none"> 如果此位为 1，则将计数器停止计数 如果此位为 0，则计数器正常工作 <p>此位上电复位默认值为 0</p>
INSTRET	2	<p>此位控制 minstret 和 minstreth 对应的计数器：</p> <ul style="list-style-type: none"> 如果此位为 1，则将计数器停止计数 如果此位为 0，则计数器正常工作 <p>此位上电复位默认值为 0</p>
Reserved	3~31	其他未使用的域为常数 0

3. SoC总线: ICB-Internal Chip Bus, 内核+Soc总线; ---- (信号有点怪啊, 他这里ready代表相对应接收到数据)

1. 特点: AXI

- 简单, 仅有两个独立通道-读写共用地址通道, 公用结果返回通道;
- 地址、数据分离;
- 地址区间寻址, 支持任意的主从数目;
- 支持地址非对齐的数据访问, 字节掩码 (Write Mask) 控制写操作;
- 支持多个滞外交易 (Multiple Outstanding Transaction)

2. 特点: AHB

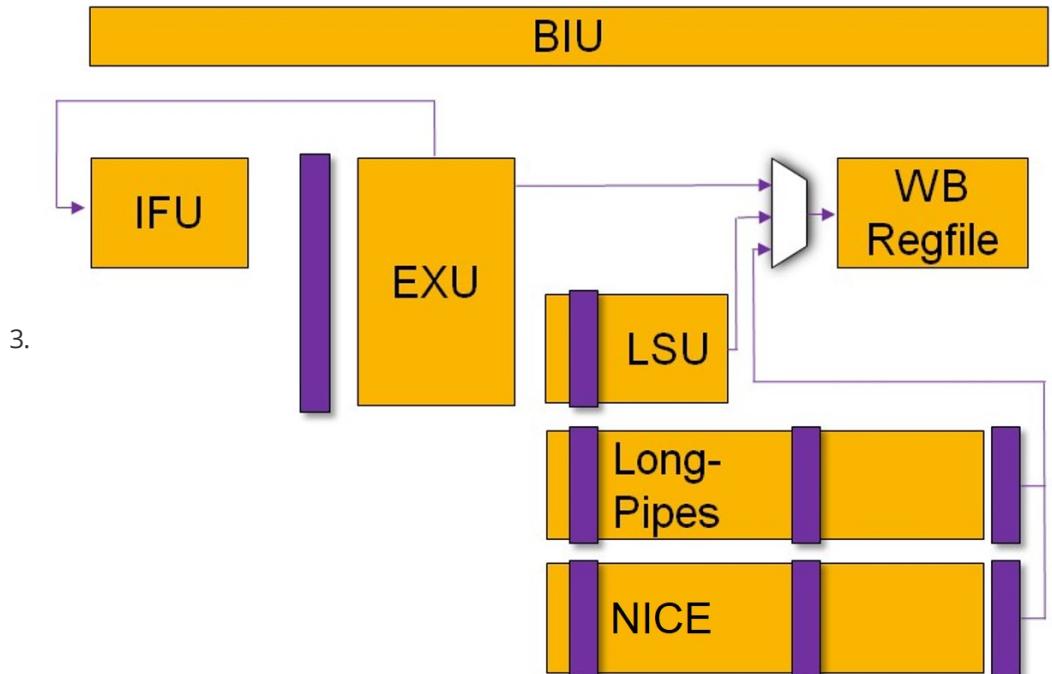
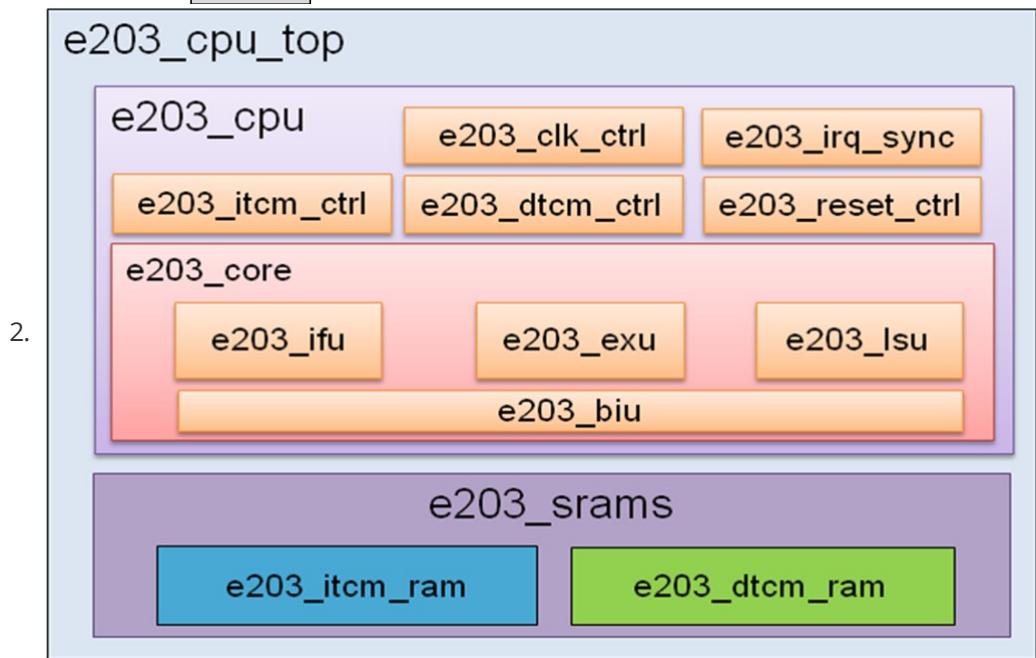
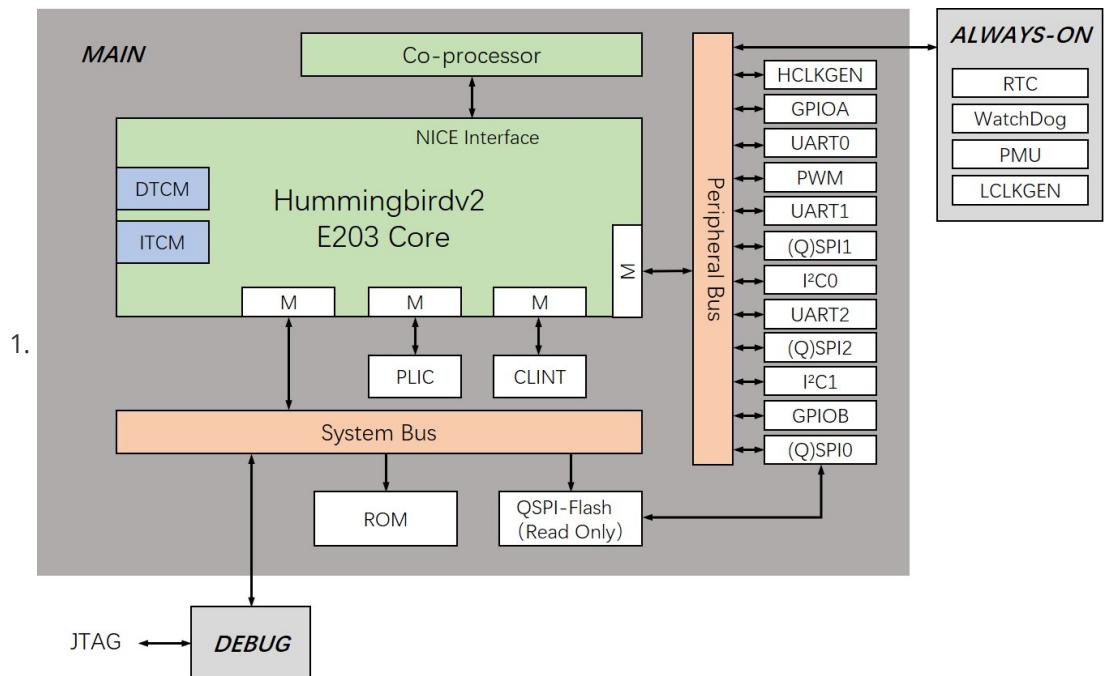
- 每个读/写都会在地址通道上产生地址;
- 不支持乱序返回、乱序完成, 返回通道数据顺序返回结果;

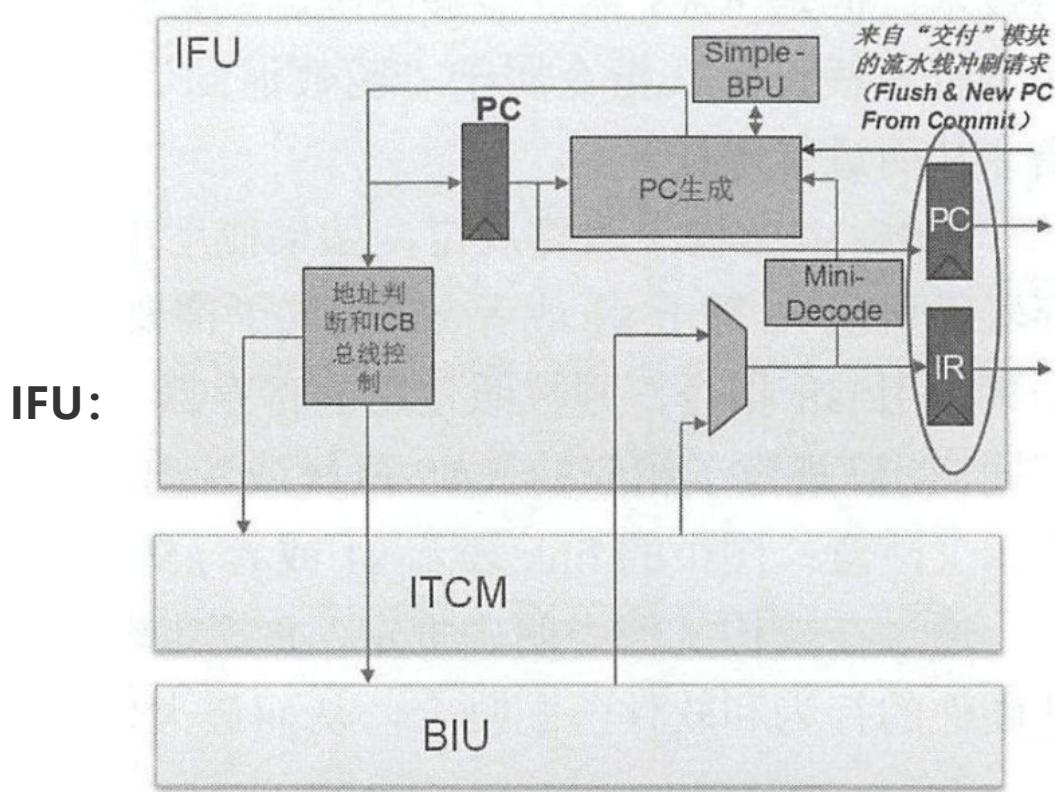
3. 协议信号:

通道	方向	宽度	信号名	介绍
Command Channel	Output	1	icb_cmd_valid	主设备向从设备发送读写请求信号
	Input	1	icb_cmd_ready	从设备向主设备返回读写接受信号
	Output	32	icb_cmd_addr	读写地址
	Output	1	icb_cmd_read	读或是写操作的指示。读则以总线宽度 (譬如 32 位) 为单位读回一个数据。写则靠字节掩码 (icb_cmd_wmask) 控制写数据的大小 (Size)。
	Output	32	icb_cmd_wdata	写操作的数据, 数据的摆放格式与 AXI 协议一致
	Output	4	icb_cmd_wmask	写操作的字节掩码, 掩码的摆放格式与 AXI 协议一致
Reponse Channel	Input	1	icb_rsp_valid	从设备向主设备发送读写反馈请求信号
	Output	1	icb_rsp_ready	主设备向从设备返回读写反馈接受信号
	Input	32	icb_rsp_rdata	读反馈的数据, 数据的摆放格式与 AXI 协议一致
	Input	1	icb_rsp_err	读或者写反馈的错误标志

4. 时序: 见文档;

4. SOC框图:





1. 分支预测：静态，向后跳转为真；

- e203_exu_oitf: Outstanding Instructions Track FIFO
 - 功能：检测出与长指令的RAW和WAW相关性；
 - fifo，深度为2表项，存储已派遣且尚未写回的长指令信息；
 - 流水线的派遣（Dispatch）点，每次派遣一个长指令，则会在 OITF 中分配一个表项（Entry），在这个表项中会存储该长指令的源操作数寄存器索引和结果寄存器索引；
- bpu: 静态，向后跳转，向前不跳；
 - 跳转：默认JAL/JALR跳转，b*** 只有立即数最高位为1（负数，向后跳转）
 - JALR特殊处理：见文档
 - X0: ;
 - X1: ;
 - 其他: ;

IFU取指过程：

1. 将预测地址低16位写入 ifuitcm 中取指，将会在下个时钟上升沿得到 64 bits 数据；

1. 这里如果是32位指令，addr需要四字节掩码对齐；如果压缩指令，就会直接取最低16位，其余部分存入缓存中；
 2. 其中内部含有一个缓存Buffer，参考蜂鸟7.3.5访问ITCM和BIU；
2. 疑问：
1. 若将AXI以外设接入，取指都是经过ITCM，但ITCM大小有限，遇到ITCM之外的指令呢？
 1. 那么取指是不经过ITCM接口，而是用过ifu2biu接口，访问总线呢？还是让ITCM访问总线呢？
 2. ITCM中保存指令是通过上电Flash冲刷保存数据，但其中保存数据以是固定的。

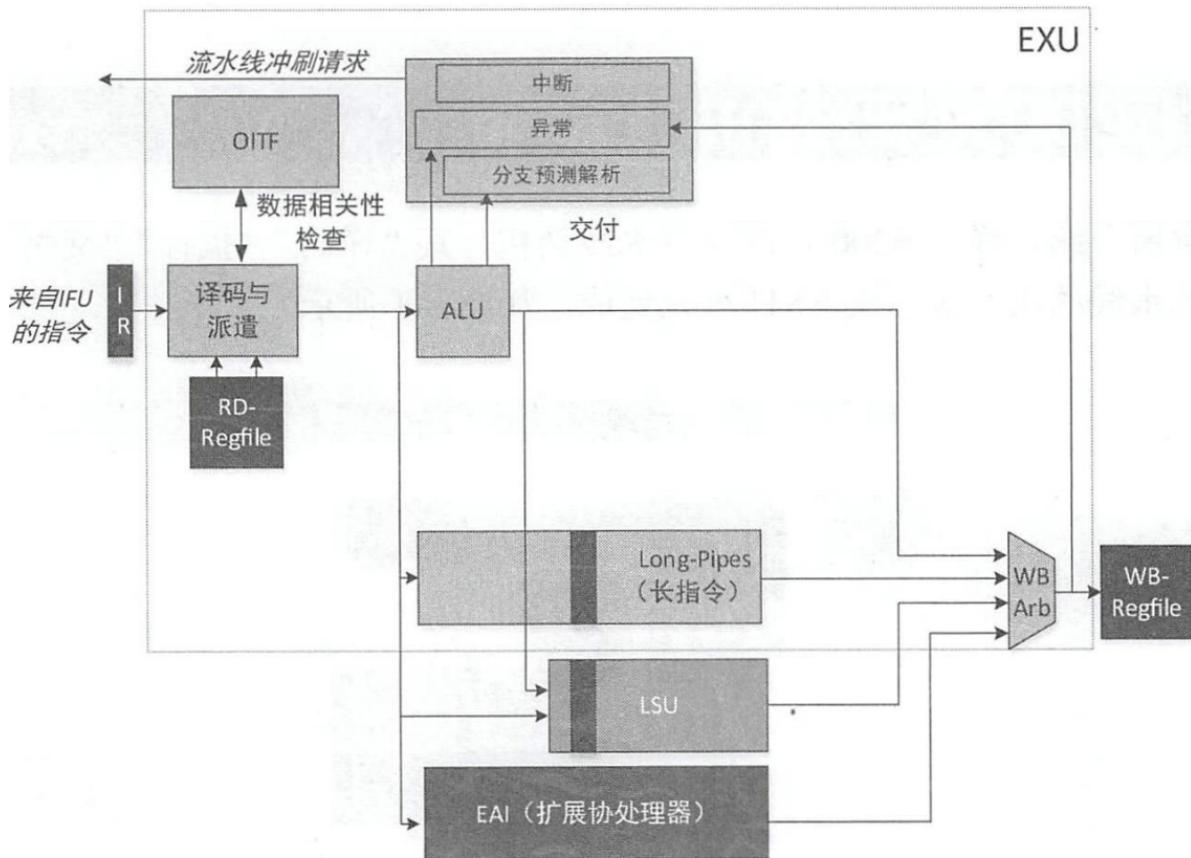
ift2icb;

1. ifu2biu_icb_cmd_valid:

1. ifu_icb_cmd_valid: ;

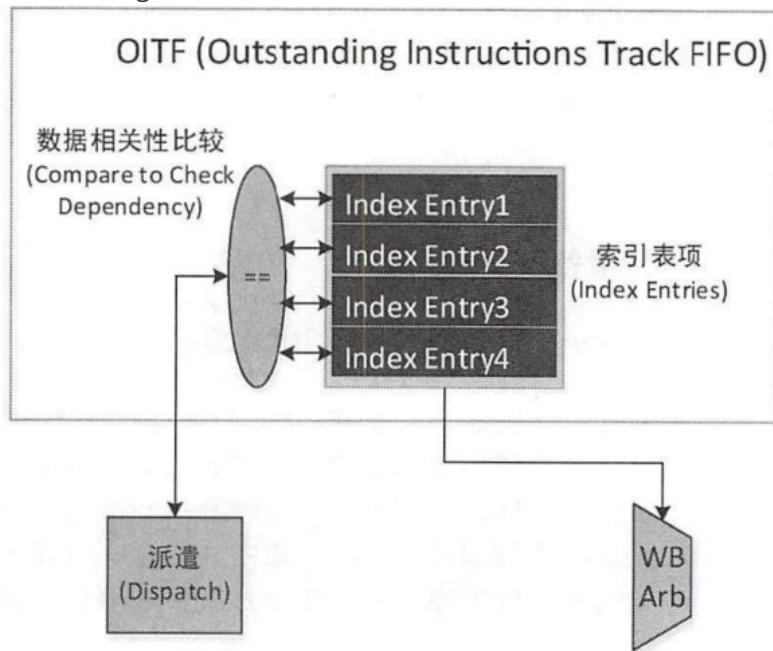
2. ifu_icb_cmd2biu: 当有ITCM时, 0x8000处地址都在ITCM寻址, 其余地址触发ICB总线

EXU:

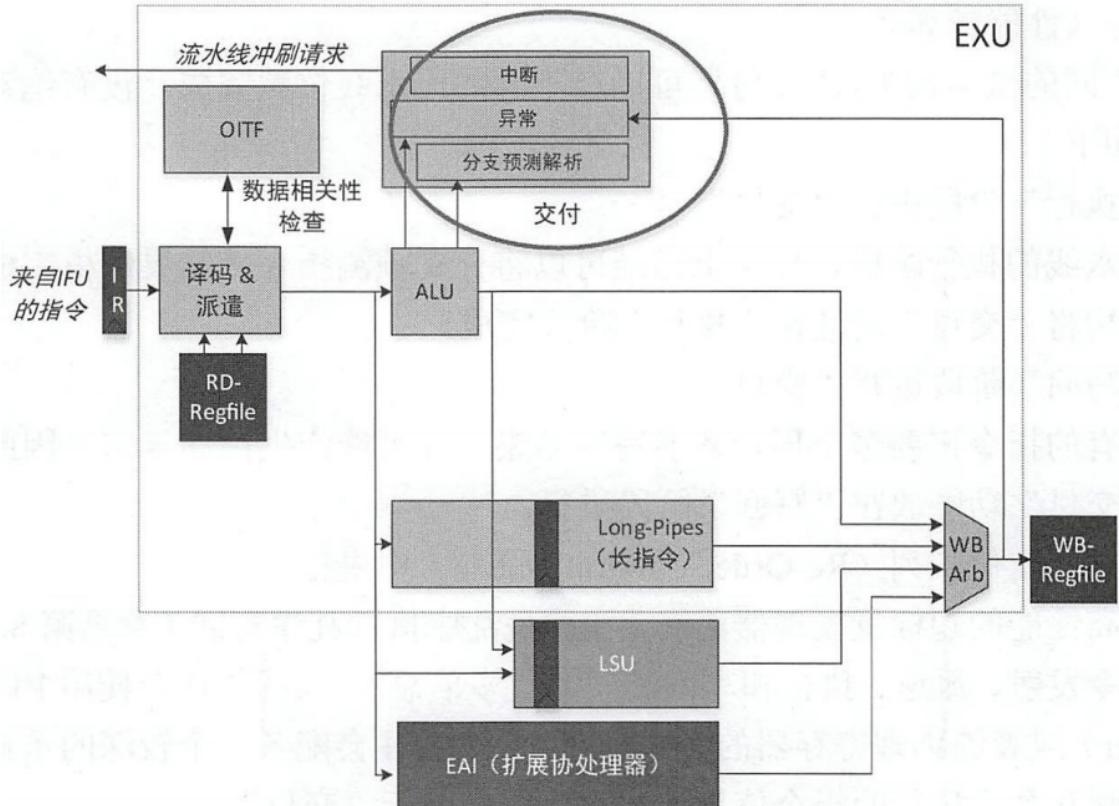


OITF:

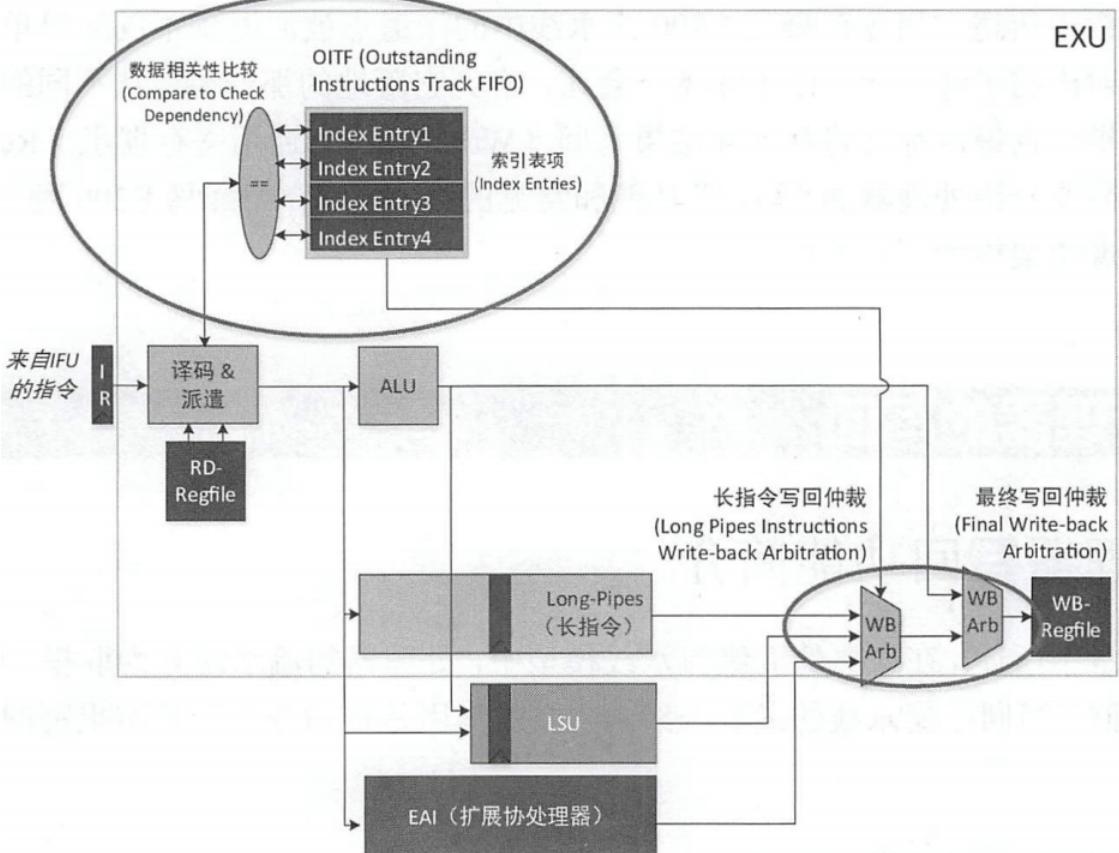
- Outstanding Instructions Track FIFO, 深度为2表项的FIFO;



WB:



- 只要前序指令没发生“分支预测错误”、“中断”、“异常”就成功交付；



MEM:

- AGU: Address Generation Unit
 - Load、store、“A”扩展指令的地址生成，以及“A”扩展指令的微操作拆分和执行；
 - 整个存储器访问指令执行的一个小环节！！！！---- 总线用得到
 - 含有ICB接口
- LSU: Load Store Unit ---参考11.4.3 P191
 - ICB总线：输入AGU、EAI；输出：BIU、DTCM、ITCM；
 - 学习知识点：ICB汇合、ICB分发--第12章

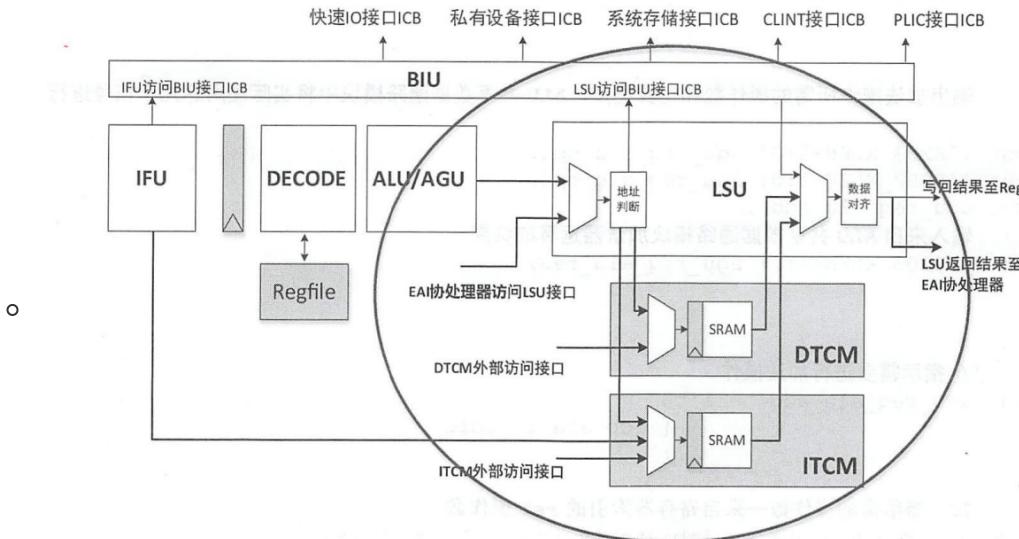


图 11-3 蜂鸟 E200 处理器核的存储器子系统结构示意图

- ITCM:

6.1 ITCM

ITCM 为 RISC-V Core 私有的指令存储器，其特性如下：

- 大小为 64KB。
- ITCM 数据宽度为 64 位。
- ITCM SRAM 虽然主要用于存放指令，但是其地址区间也可以被 Load、Store 指令访问，从而用来存放数据。

6.2 DTCM

DTCM 为 RISC-V Core 私有的数据存储器，其特性如下：

- 大小为 64KB。
- DTCM SRAM 数据宽度为 32 位。

- 若将AXI4以外设接入ICB模块，其中在蜂鸟取指处理中只有0x8000_xxxx是映射itcm，其余都是映射到BIU。其中在tb文件中是将测试文件中的[0: 0x0001_0000]保存在ITCM中。
 - 遇到ITCM映射之外的指令----从外部存储器中读取，IFU通过BIU使用系统存储接口访问外部的存储器。比如系统上电后的引导程序可能从外部Flash中读取)
 - ITCM就是映射一部分内存，所以当Store存储地址恰好在ITCM区间就会触发LSU；
- sram接口信号：ds、sd低电平，ls为cpu发出信号，目前还不知道功能（cpu->ITF-- ICB Interface from Ifetch应该为clk_ctrl）；
- ICB总线接口：
 - 一组输入：数据宽度32位，LSU访问，用于上电存储数据；

- 两组输入：数据宽度 64 位的 IFU 专用ICB 接口，数据宽度 32 位的外部直接访问（外部接口，便于其他外设访问）；
- 三组总线汇总为一组ICB总线，优先级：LSU > 外部接口> IFU；
- 64位单口SRAM，可配置大小（LSU、ITCM位宽位32位，需要数据转换）；
- 模块原理：
 - 先LSU、ITCM外设 数据位宽通过实例化 **sirv_gnrl_icb_n2W** 扩展到64位（ext2itcm -> ext）；
 - 然后将二者将其控制信号拼接为位宽*2的信号 ({ext, lsu} ->arbt_bus) , 通过实例化 **sirv_gnrl_icb_arbt** 仲裁（主要是仲裁LSU与ext优先级, arbt_bus -> arbt）；
 - 然后将仲裁arbt信号与ifu选择通过优先级 (IF > LSU > 外设) 控制，生成icb接口信号；
 - 通过实例化 **sirv_sram_icb_ctrl** 模块将icb接口信号转换为sram接口信号以及cmd、rsp 反馈信号；
 - 内部实例化 **sirv_gnrl_bypbuf** 模块，将cmd 输入信号缓存一周期，防止ready反压；
 - 实例化 **sirv_1cyc_sram_ctrl** 模块，将缓存icb总线信号处理生成相对应sram信号以及反馈信号；

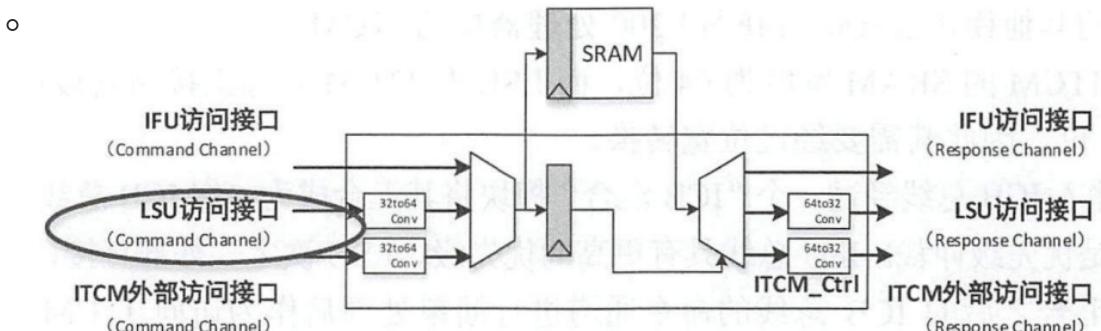


图 11-4 ITCM 微架构示意图

- DTCM:

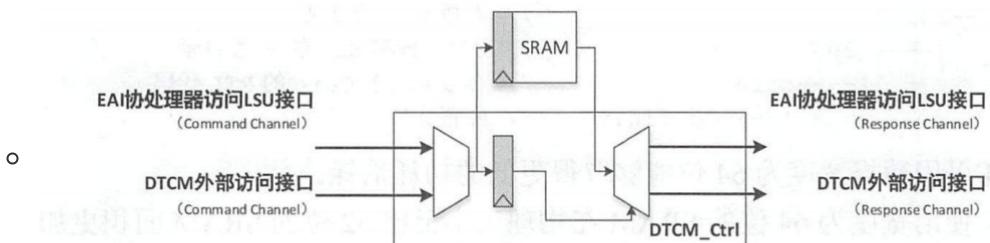


图 11-5 DTCM 微架构示意图

- 32位单口SRAM；
- 两组ICB输入总线，LSU、外部直接访问接口，同理汇总为一条总线，LSU>外部接口；
- "A"扩展指令处理：在多线程情形下访问存储器的原子（Atomic）操作或者同步操作
 - Load-Reserved和Store-Conditonal: LSU设置互斥检测器（EXclusive Monitor）；
 - AMO (Atomic Memory Operation) 指令；
- Fence指令：
 - RISC-V架构采用松散存储器模型，松散存储器模型对于访问不同地址的存储器读写指令的执行顺序不作要求，除非使用明确的存储器屏障指令（Fence、Fence.l：用于强行界定存储器访问的顺序）；
 - 在程序中，如果添加了 Fence 指令，则 Fence 指令能够保证“在 Fence 之前所有指令造成的访存结果”必须比“在 Fence 之后所有指令造成的访存结果”先被观测到。
 - 在程序中，如果添加了 Fence .l 令，则“在 Fence.l 之后所有指令的取指令操作”定能够观测到“在 Fence.l 之前所有指令造成的访存结果”；

- 处理：蜂鸟——fence当成 fence iorw, iorw指令实现；在流水线派遣点，必须等待所有已经滞外的指令执行完毕；

ICB：

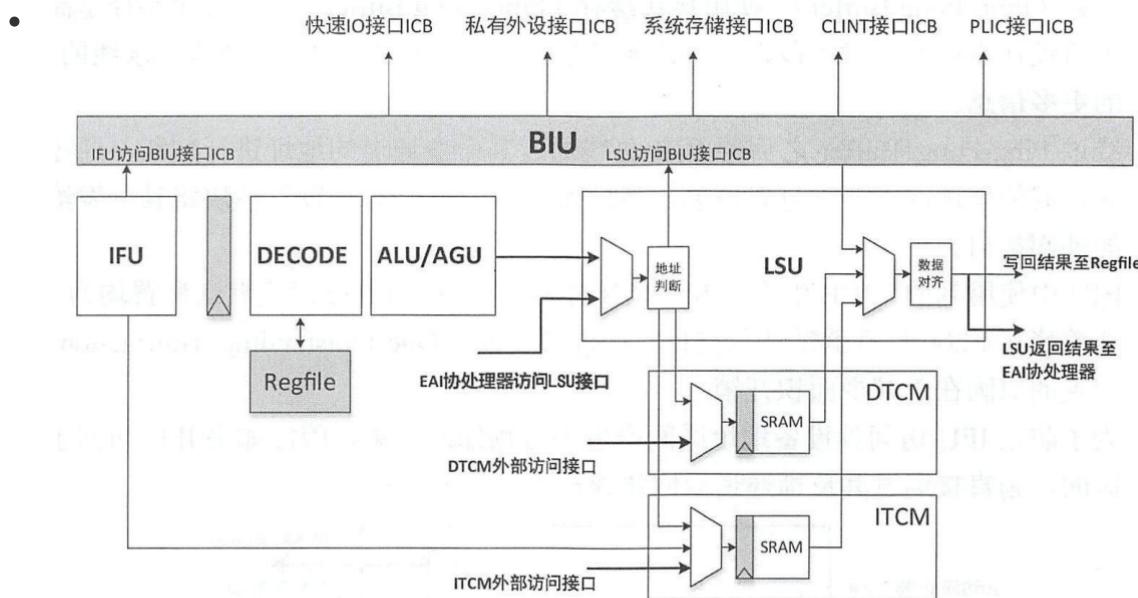


图 12-14 BIU 在蜂鸟 E200 处理器核中的位置

- fifo：支持滞外交易，主要用于仲裁输出反馈信号；

表 12-1

ICB 总线信号

通道	方向	宽度	信号名	介绍
命令通道	Output	1	icb_cmd_valid	主设备向从设备发送读写请求信号
	Input	1	icb_cmd_ready	从设备向主设备返回读写接受信号
	Output	DW	icb_cmd_addr	读写地址
	Output	1	icb_cmd_read	读或是写操作的指示
	Output	DW	icb_cmd_wdata	写操作的数据
	Output	DW/8	icb_cmd_wmask	写操作的字节掩码
反馈通道	Input	1	icb_rsp_valid	从设备向主设备发送读写反馈请求信号
	Output	1	icb_rsp_ready	主设备向从设备返回读写反馈接受信号
	Input	DW	icb_rsp_rdata	读反馈的数据
	Input	1	icb_rsp_err	读或者写反馈的错误标志

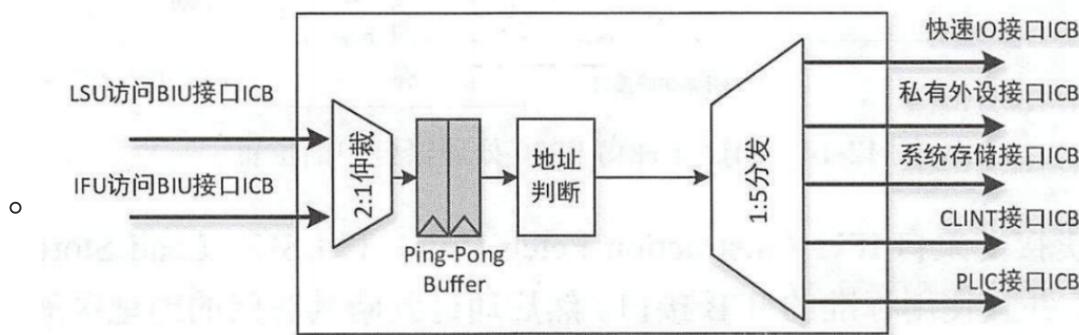


图 12-15 BIU 微架构图

ICB2AXI:

- 注意：ICB不支持brust

EAI：16章

EAI 指令的编码

32位的EAI指令编码格式如图16-1所示。

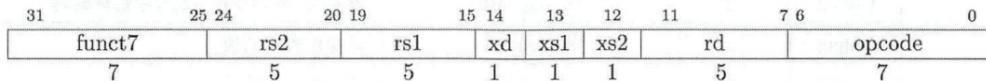


图 16-1 EAI 指令编码格式

- (1) 指令的第0位至第6位区间为Opcode编码段，根据表16-1中的编码规则使用编码ym-0、custom-1、custom-2和custom-3指令组。
- (2) xs1、xs2和xd比特位分别用于控制是否需要读源寄存器rs1、rs2和写目标寄存器rd。
 - 如果xs1位的值为1，则表示该指令需要读取由rs1比特位索引的通用寄存器作为源
 - 滞外指令**：Custom指令从被发送至协处理器到协处理器反馈并退役之间的时间；(不超过4条)
 - 读写存储器优先级：协处理器>主处理器指令

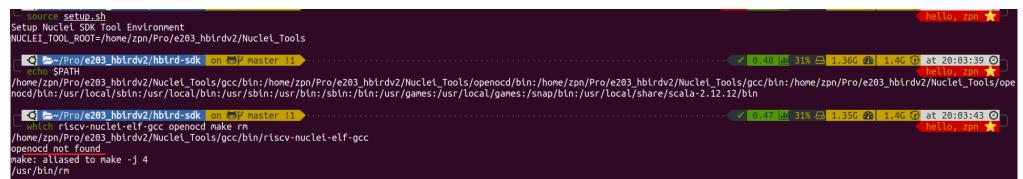
二、BUG：

- 安装E203 sdk遇到问题：

- 参考：

- https://doc.nucleisys.com/hbirdv2/quick_start/sdk.html#
 - https://mp.weixin.qq.com/s?_biz=MzkzNjI4NDlwNg==&mid=2247485410&idx=1&s_n=333a6ec4ba2a1837e0f0ee5a8f77e3ff&chksm=c2a05ea3f5d7d7b5b0acf1fc8b30d3022f80d63c6f5962239c3a385d831c59edfc0c3097b759&token=1326539848&lang=zh_CN#rd

- 问题：在linux下安装sdk，执行`which riscv-nuclei-elf-gcc openocd make rm`时找不到openocd；



```
source Setup.sh
Setup Nuclei SDK Tool Environment
NUCLEI_TOOL_ROOT=/home/zpn/Pro/e203_hbirdv2/Nuclei_Tools
└─[1] ➜ /home/zpn/Pro/e203_hbirdv2/hbirdv2/Nuclei_Tools/openocd/bin:/home/zpn/Pro/e203_hbirdv2/Nuclei_Tools/gcc/bin:/home/zpn/Pro/e203_hbirdv2/Nuclei_Tools/openocd/bin:/home/zpn/Pro/e203_hbirdv2/Nuclei_Tools/gcc/bin:/home/zpn/Pro/e203_hbirdv2/Nuclei_Tools/openocd/bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/usr/games:/snap/bin:/usr/local/share/scala-2.12.12/bin
$ which riscv-nuclei-elf-gcc openocd make rm
/home/zpn/Pro/e203_hbirdv2/Nuclei_Tools/openocd/bin/riscv-nuclei-elf-gcc
openocd not found
make: allased to make -j 4
/usr/bin/rm
```

- 原因：下载的openocd文件中bin文件夹是在2022.12中，而执行脚本并不包含这个文件；

- 修改：将2022.12文件移动到上一层文件当中，

```
q ~/Pro/e203_hbirdv2/Nuclei_Tools on main ?2
tree -L 2

-
  -- gcc
    |-- bin
    |-- build.txt
    |-- gitrepo.txt
    |-- include
    |-- lib
    |-- lib64
    |-- libexec
    |-- riscv-nuclei-elf
    |-- share
  -- openocd
    |-- bin
    |-- contrib
    |-- distro-info
    |-- OpenULINK
    |-- README.md
    |-- scripts
    |-- share

15 directories, 3 files
```