Riscv 的基本原理无须赘述，本论文改写的 CV32E40P 是一种基于 RI5CY 的，适用于 pulpino 平台的 4 级 RISC-V 流水线 CPU。

首先要做的当然是对原本的内核进行测试，以便与改写后的内核进行测试。

以下模块的基本作用是来自于网上的资料，部分有出入但总体差别不大。

各模块作用如下：

alu.sv 算术逻辑运算单元，实现了大部分算术逻辑运算

alu_div.sv实现了除法运算

compressed_decoder.sv 将 16bit 的压缩指令扩展为等价的 32bit 指令

controller.sv 实现了流水线的控制通路

cs_registers.sv 实现了 CSR

debug_unit.sv 调试单元

decoder.sv 实现对指令的译码

ex_stage.sv 对应流水线的执行阶段

exc_controller.sv 异常控制器

hwloop_controller.sv 硬件循环控制器

hwloop_regs.sv 硬件循环对应的寄存器

id_stage.sv 对应流水线的译码阶段

if_stage.sv 对应流水线的取指阶段

load_store_unit.sv 实现了对数据存储器的访问

mult.sv 实现了整数乘法、点积运算

prefetch_buffer.sv 实现了指令预取单元，并且是配置一：可以存放 3 条 32 位指令，按照 FIFO 原则使用

prefetch_L0_buffer.sv 实现了指令预取单元，并且是配置二：大小等于指令缓存 line，即 128 位

register_file.sv实现了寄存器文件，32 个 32 位寄存器，具有 3 个读端口，2 个写端口，并且采用的锁存器实现的，目标是针对 ASIC 应用

register_file_ff.sv 也实现了寄存器文件，32 个 32 位寄存器，具有 3 个读端口，2 个写端口，但是采用的触发器实现的，目标是针对 FPGA 应用

riscv_core.sv RI5CY 的顶层模块

riscv_simchecker.sv 仿真过程检验

riscv_tracer.sv 记录执行过的指令

同时，想要对核进行测试的话，也需要创建一个 PULPino 项目，并把核嵌入进去，因此，想要进行进一步的工作的话，首先还要对 PULPino 进行学习。

PULPino 中的顶层模块是 pulpino_top,与核，也就是 RI5CY 联系最紧密的当属 core_region 这个中层模块（自己分析，可能在之后的学习中被否定）。

经过寻找和对比，在 core_region 模块里面找到了一个 RISCV_CORE 的模块调用，而且对应的调用模块不存在，假设，这就是要寻找的对应的 RI5CY 应在的位置。

然后，为了进一步确认是不是我们要找的东西，将其接口和 RI5CY 顶层模块 riscv_core 的接口进行对比。

经过对比发现二者基本上是可以对的上号的，除了少部分接口包括：

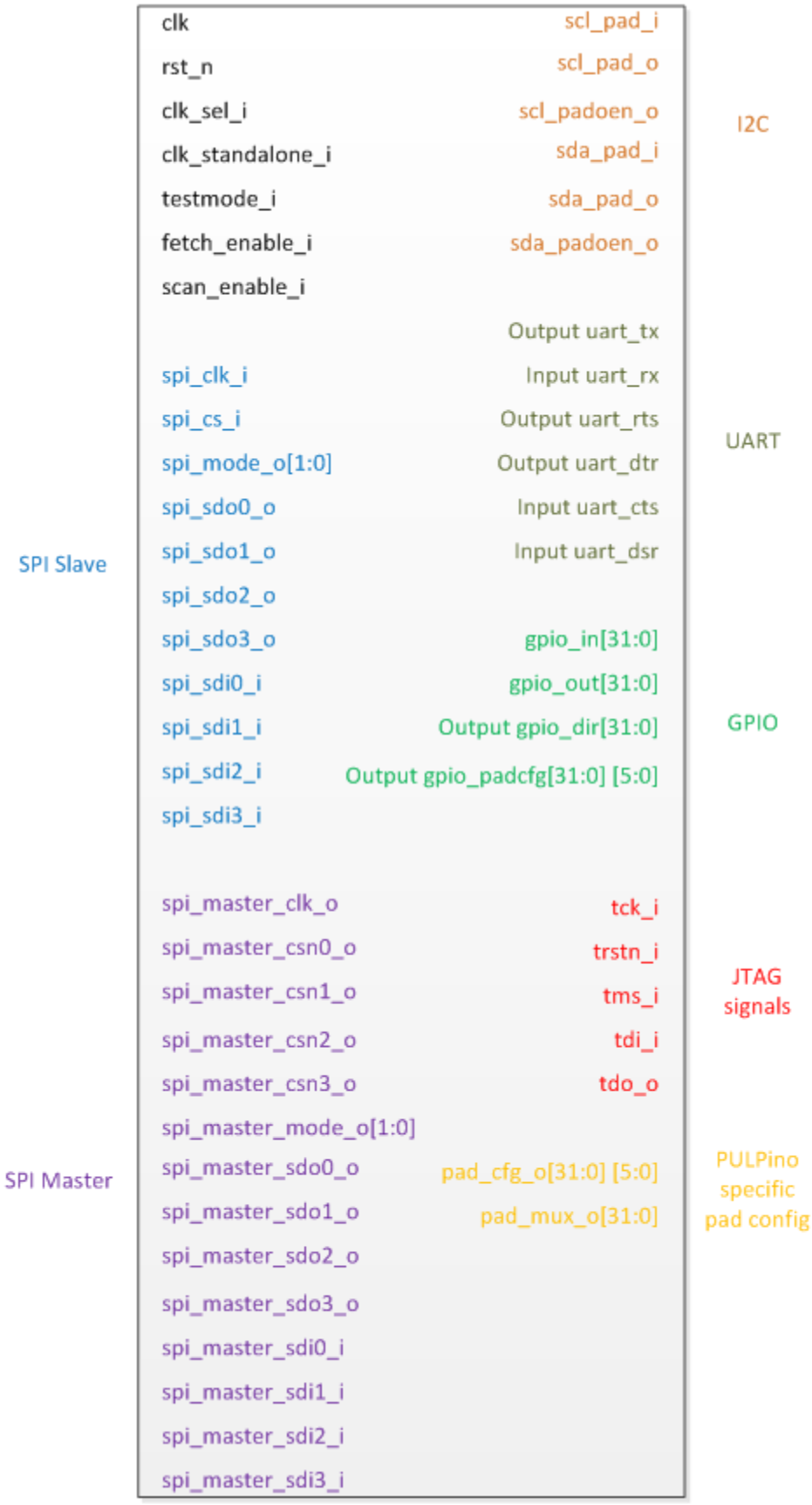
```
input    logic            fregfile_disable_i,    // disable the fp regfile, using int regfile instead
//riscv_core 有但是 core_region 没有
            .irq_i                ( (|irq_i)              ),
            .irq_id_i             ( irq_id                ),
```

```
            .irq_ack_o          (                       ),
            .irq_id_o           (                       ),
```
//core_region 的

```
// Interrupt inputs
   output logic             irq_ack_o,
   output logic [4:0]    irq_id_o,
   input    logic             irq_sec_i,

   input    logic             irq_software_i,
   input    logic             irq_timer_i,
   input    logic             irq_external_i,
   input    logic [14:0] irq_fast_i,
   input    logic             irq_nmi_i,
   input    logic [31:0] irq_fastx_i,
```
//riscv_core 的
以上思路不对
在下面又看到了一个 RISCV_CORE，经过对比，这个才应该是对应 RI5CY 的模块调用。
然后就是创建一个对应的模块调用，把 RI5CY 的代码加入到创建的 PULPino 项目里面。
此时 USE_ZERO_RISCY 信号应该置 0。（后话）
此时，riscv_core 里面有但是 rsicv_region 里面没有的接口有：

```
   input    logic             fregfile_disable_i,   //禁用浮点 regfile，改为使用整型 regfile
```
接下来我们回头来看 PULPino_top 里面的输入输出接口，在网上找了个图，如下：
其中的信号依据下图可以分类：
clk　时钟信号
rst_n　　复位信号
clk_sel_i 用来选择工作时钟，如果是 0，那么时钟就是 clk，反之，时钟来自一个锁频环，用于 ASIC 生产时，clk_sel_i 设置为 1
clk_standalone_i　与锁频环相关的控制信号
testmode_i　如果为 1，那么禁止 clock gate，反之，使能 clock gate
fetch_enable_i　　如果为 1，表示开始取指译码执行
scan_enable_i 与锁频环相关的控制信号
//以上可以统归为一类全局信号接口
其他的，名字里面带 GPIO、UART、I2C、SPI master 的统统是对应的外设接口
Spi slaver



到这个时候暂时就没有头绪了，干脆点一下试着编译，果然出现错误，在 xvlog.log 里面找到错误原因如下:

ERROR: [VRFC 10-91] fpnew_pkg is not declared [E:/2020/cv32e40p-master-2/rtl/include/riscv_defines.sv:486]
ERROR: [VRFC 10-91] fpnew_pkg is not declared [E:/2020/cv32e40p-master-2/rtl/include/riscv_defines.sv:487]
ERROR: [VRFC 10-91] fpnew_pkg is not declared [E:/2020/cv32e40p-master-2/rtl/include/riscv_defines.sv:488]
ERROR: [VRFC 10-1040] module riscv_defines ignored due to previous errors [E:/2020/cv32e40p-master-2/rtl/include/riscv_defines.sv:26]
可是找遍了所有的文件夹都没发现 Fpnew_pkg.sv 这样一个文件，这个时候，返回到 github，发布 RI5CY 的发布页面给出了一个网站”https://github.com/pulp-platform/fpnew/blob/develop/src/fpnew_pkg.sv”，让去这个网站上找 fpnew_pkg.sv
找到 fpnew_pkg.sv 以后将对应的变量直接写进错误的地方（不再把 fpnew_pkg.sv 引入）
这个错误解决之后出现了第二个错误：
ERROR: [VRFC 10-2063] Module <core2axi> not found while processing module instance <core2axi_i> [E:/2020/pulpino-master-2/rtl/core2axi_wrap.sv:42]
ERROR: [VRFC 10-2063] Module <axi_slice_wrap> not found while processing module instance <axi_slice_core2axi> [E:/2020/pulpino-master-2/rtl/core_region.sv:318]
ERROR: [VRFC 10-2063] Module <instr_ram_wrap> not found while processing module instance <instr_mem> [E:/2020/pulpino-master-2/rtl/core_region.sv:391]
ERROR: [VRFC 10-2063] Module <axi_mem_if_SP_wrap> not found while processing module instance <instr_mem_axi_if> [E:/2020/pulpino-master-2/rtl/core_region.sv:409]
ERROR: [VRFC 10-2063] Module <ram_mux> not found while processing module instance <instr_ram_mux_i> [E:/2020/pulpino-master-2/rtl/core_region.sv:434]
ERROR: [VRFC 10-2063] Module <sp_ram_wrap> not found while processing module instance <data_mem> [E:/2020/pulpino-master-2/rtl/core_region.sv:476]
ERROR: [VRFC 10-2063] Module <adv_dbg_if> not found while processing module instance <adv_dbg_if_i> [E:/2020/pulpino-master-2/rtl/core_region.sv:563]
ERROR: [VRFC 10-2063] Module <peripherals> not found while processing module instance <peripherals_i> [E:/2020/pulpino-master-2/rtl/pulpino_top.sv:208]
ERROR: [VRFC 10-2063] Module <axi_node_intf_wrap> not found while processing module instance <axi_interconnect_i> [E:/2020/pulpino-master-2/rtl/pulpino_top.sv:296]
ERROR: [XSIM 43-3322] Static elaboration of top level Verilog design unit(s) in library work failed.
可以看到，这些错误大都是由于没有找到对应的模块所导致的，接下来就是漫长的 debug 过程。
首先是 mailbox 的错误，在网上只能找到一个老外的回答，可能是我用的 vivado 2015.4 不支持 mailbox，后来发现是 systemverilog 自带 use std，因此我在程序最前面加了一个 use Verilog::Std，可还是报错，待解决
接下来处理其他的报错，由于本实验的核心在于 RI5CY，因此对于 PULPino 中的一些无用模块尽量先舍弃，编译通过再想其他的事情。
其他的模块缺失里面，首先，因为 core_region 已经确定要使用 RI5CY 内核，因此另一个内核模块就可以删掉了，直接让 pulpino 连上 RI5CY 就可以了。
做到这一部发现之前的 std 怎么弄也弄不好，这时尝试下载一下 Quartus，看看行不行，Quartus Prime 安到一半卡了好久，最后还是没有安装成功，这个时候和学长老师沟通了一

下，他说不要搞 pulpino 的，直接做 RI5CY 的就可以了，那要直接做 RI5CY，那最重要的首先就是 data 和 instr 寄存器，今天先到这里，大数据还有两个实验。

今天的主要任务就是要具体地看一下 RI5CY 的外层接口，看一下仿真需要什么样的接口，就前面所讲的，pulpino 连到 RI5CY 上面有两个寄存器，这两个寄存器应当是相当重要的。

首先当然是最重要的，要把 bug 全部搞定，昨天碰到个很致命的 bug，vivado2015.4 不支持 mailbox 数据结构，这个在网上唯一搜索到的相关讨论是在一个国外论坛里面，有一位网友提出了自己的 vivado2015.4 无法运行含有 mailbox 的 sv 代码，下面一个比较靠谱的答案的回答是没有 std 库，要想办法找到对应的 std 库，然后把库导入到这个项目里。

除此之外还有很重要的一点就是写测试用的 RISCV 代码，RISCV 指令测试主要有两个难点，一个是跳转指令，还有一个是内存空间操作指令。这两个都涉及到存储器的操作，所以比较麻烦。想要解决这个麻烦，正确地完成仿真，一个方法是直接用 PULPino 的源码，但这个方法太复杂太难，第二个方法是自己写两个存储器（也可以尝试用 pulpino 里面的代码来改写）。

那么接下来把之前写的核接口再拿出来，等下再分析。

导入后出现的第一个错误是没有找到 fpnew_top 模块，根据名字可以猜测，这个应该是一个有关于浮点操作的模块，尽管一开始没头绪，但后来找到了 github 上面的发布页里面，找到了可以下载这个模块的分享页。尝试对比接口。接口基本上一样。

将 fpnew 的代码导入项目，发现出现了一大堆错误。尝试 debug。

需要模块 common_cells，同样在 github 上面找到。

同样地导入后发现有六个模块出现了 include 语句的错误，导入后也无法修复。观察后发现将`include "common_cells/registers.svh"改成`include " registers.svh"就好了。

据观察以上的东西都是为了浮点运算做工作，同时根据 github 上面的发布名，它们都是 pulp 的辅助工具。

以上步骤都完成了，最后发现最后一个 bug 就是昨天整出来的“mailbox”想要解决这个东西唯一的方法就是导入 Verilog::Std，但昨天一直都没有找到方法。

这个时候发现 riscv_tracer 只有输入没有输出，这就说明这个模块实际上对总体没什么影响，意思就是即是删去了也不会对整个流水线产生什么影响。于是我就将它删去了，剩下的东西再编译一次，mailbox 错误不出预料地消失了。但是又多出来几个错误：

[XSIM 43-3209] "E:/2020/cv32e40p-master-4/rtl/riscv_decoder.sv" Line 1741. Unsupported construct.

[XSIM 43-3209] "E:/2020/cv32e40p-master-4/rtl/riscv_decoder.sv" Line 1741. Unsupported construct.

[XSIM 43-3209] "E:/2020/cv32e40p-master-4/rtl/riscv_decoder.sv" Line 1870. Unsupported construct.

[XSIM 43-3209] "E:/2020/cv32e40p-master-4/rtl/riscv_decoder.sv" Line 1870. Unsupported construct.

[XSIM 43-3209] "E:/2020/cv32e40p-master-4/rtl/riscv_decoder.sv" Line 1875. Unsupported construct.

[XSIM 43-3209] "E:/2020/cv32e40p-master-4/rtl/riscv_decoder.sv" Line 1875. Unsupported construct.

到对应行去看一下可发现可能是 vivado 不支持 sv 语言中的某种语法或者句式。

这里碰到了一个疑点，1741 行的代码格式几乎和 1748 行一模一样，但是 1748 就没有报错。这个时候先改一下试试看。

我将比较简单的代码：

```
unique case (frm_i) inside
                    [3'b000:3'b100] : fp_rnd_mode_o = frm_i; //legal rounding modes
                    default         : illegal_insn_o = 1'b1;
                endcase
```

改成了比较冗余的：

```
unique case (frm_i) inside
                    3'b000:fp_rnd_mode_o = frm_i;
                    3'b001:fp_rnd_mode_o = frm_i;
                    3'b010:fp_rnd_mode_o = frm_i;
                    3'b011:fp_rnd_mode_o = frm_i;
                    3'b100:fp_rnd_mode_o = frm_i;
                    default         : illegal_insn_o = 1'b1;
                endcase
```

这一处 bug 就被修正了，接下来是下面两处

全部修正完成，仿真通过，不知前面的几个 bug 是不是 vivado 的 bug。

这样运行就没有 bug 了，接下来就是先考虑怎么对这个内核进行测试了。

在此之前，本科生有一项大作业就是做一个 cpu，那个 cpu 用的也是哈佛结构，pulpino 用的也是哈佛结构。像之前说的，要想比较好的运行，不能只考虑算术指令的仿真，还要考虑其他指令，包括跳转指令、存取指令，也要能够被很好的运行。

尽管初步的 debug 完成，但是这个时候我们还是要继续深入的考虑下去的话，像前面说的，必须要完成寄存器模块。

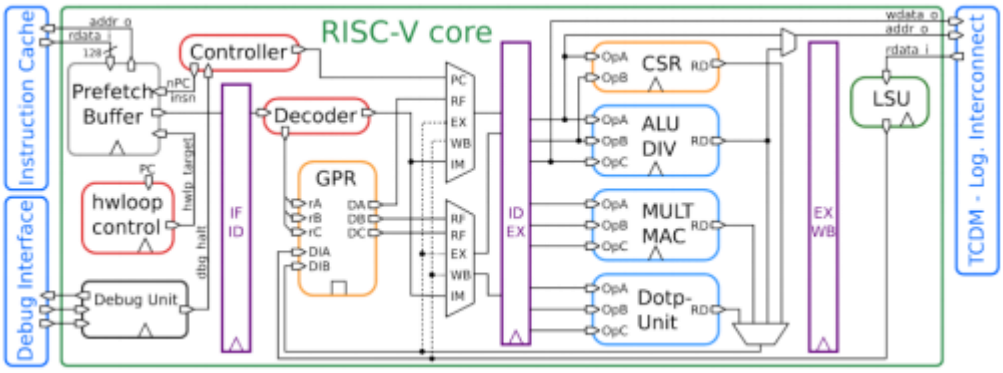
RI5CY 的源代码里面最可能产生与寄存器链接的莫过于 riscv_load_store_unit，除此之外，里面还有几个和顶层模块 riscv-core 平行的模块，也很有可能是存储器的关键部分------riscv_register_file_latch 模块。

下面阅读这两个模块的原码。

阅读了一会儿发现模块 riscv_register_file_latch 并不是存储器，而只是一个 16 位的寄存器。

看 riscv_load_store_unit，这个才应该和要找的存储单元有关。

阅读了一下发现有点困难，好多地方还是很迷糊，决定结合网上的资料整体地看一遍。



这是 RI5CY 的整体的大致电路图（源码里面一个链接上的）。

首先这个内核是一个四级流水线结构，在源码里面，它已经将自己的 if，id，ex 功能组件分别装进了 riscv_if_stage，riscv_id_stage，riscv_ex_stage，每个 stage 内部都有若干功能模

块，这些模块都在一定程度上是并行的。

对于 if 部分。

主要有三个部分：

prefetch_buffer（包括 riscv_prefetch_buffer 和 riscv_prefetch_L0_buffer，这两个看了下源码，不带 L0 的数据是 32 位的，带 L0 的数据是 128 位的，决定用哪一个模块由 RDATA_WIDTH 决定,默认是 32 位），这个模块的功能主要是以下从网上复制过来的：指令预取 Buffer 位于处理器核与共享的指令缓存之间，共享的指令缓存可能会由于多个处理器核同时访问，而增加延迟，为此，为每个处理器设计了一个容量很小的指令预取 Buffer，可以在不影响面积的前提下提高性能。除此之外，还有一个理由，RI5CY 支持 RV32C，即指令可能是 16 位的，此时取指的地址可能不是 4 字节对齐，这种非对齐访问，至少需要两个时钟周期才能取得指令，通过增加指令预取 Buffer，能够实现一个时钟周期取得指令。指令预取 Buffer 的大小可以有两种配置：配置一：可以存放 3 条 32 位指令，按照 FIFO 原则使用。配置二：是指令缓存 line 的大小，比如 128 位。需要注意一点，对于配置二而言，实际大小并不是 128 位，而是 128+32 位，这主要是考虑到有些指令会跨两条指令缓存 line。在指令缓存中，上一条 line 的最后 32bit 的低 16bit 是一个 16 位的压缩指令，所以在取下一条指令缓存 line 的时候，需要暂时保存上一条 line 的最后 32 位，其中的高 16bit 与下一条指令缓存 line 的低 16bit 组成一条完整的指令。这也解释了为何通过添加指令预取 Buffer 可以实现即使指令地址不是 4 字节对齐的，也可以在一个时钟周期取得指令的原因。

网上只给出了两个模块的内容，可想而知这两个模块的内容必然是相对来讲非常重要的，另一个被提到的模块是 load_store_unit，这个模块的主要任务是 RI5CY 与数据存储器的桥梁，如果是这样的话，这个模块就是接下来要做测试的关键了。

首先看看顶层模块是怎么调用这个模块的，就是非常正常的调用，源码甚至在这个模块调用前面加了一个大大的花名。看一个模块，首先就要看接口，在网络上找到了 lsu 和外部存储器链接的接口如下：

data_req_o	输出	请求有效信号，在一次访问中，该信号保持为 1，直到输入信号 data_gnt_i 持续一个时钟周期为 1，此时表示本次访问结束
data_addr_o[31:0]	输出	访问地址
data_we_o	输出	为 1，表示是写操作，反之，表示是读操作
data_be_o[3:0]	输出	以字节为粒度的使能标志，每一 bit 对应一个字节
data_wdata_o[31:0]	输出	要写的数据
data_rdata_i[31:0]	输入	从数据存储器返回的数据
data_rvalid_i	输入	为 1，表示数据访问请求处理完毕
data_gnt_i	输入	为 1，表示数据存储器一侧接收了本次访问请求，此时，不一定给出访问结果，但是 LSU 侧可以继续访问下一个地址

验证一下，看一看对应的顶层模块的接口是不是要求连接到存储器的接口。经过观察发现 output logic [31:0] data_wdata_o,和 input logic [31:0] data_rdata_i,模块是直接连接到 riscv_core 顶层模块的对外接口上的。由于直接连到对外接口，这些接口在 lsu 里的名字就是它们在 riscv_core 上面的名字。

在顶层接口我们找到了 // Data memory interface

```
output logic          data_req_o,
input  logic          data_gnt_i,
input  logic          data_rvalid_i,
output logic          data_we_o,
output logic [3:0]    data_be_o,
```

```
output logic [31:0] data_addr_o,
output logic [31:0] data_wdata_o,
input  logic [31:0] data_rdata_i,
output logic [5:0]  data_atop_o, // atomic operation, only active if parameter `A_EXTENSION != 0`
```

显然这就是我们要写的 mem 与 RI5CY 核心的连接接口。

（其实在这个地方，只要一开始是从 riscv_core 开始看的话根本不需要浪费这么大波折，直接看就可以了）

由于 pulpino 是哈佛结构，同样的，直接在顶层找，找指令存储器的接口。按道理就在数据内存的上面，果然我们找到了：// Instruction memory interface

```
output logic                         instr_req_o,
input  logic                         instr_gnt_i,
input  logic                         instr_rvalid_i,
output logic [31:0]                  instr_addr_o,
input  logic [INSTR_RDATA_WIDTH-1:0] instr_rdata_i,
```

按照以上的方法，再看一次 riscv_core 的对外接口：

这里只列出注释：

```
// Clock and Reset，无需赘述，时钟和重启信号
// Core ID, Cluster ID and boot address are considered more or less static
//data mem
//instruction mem
// handshake signals
// request channel
// response channel
// Interrupt inputs
// Debug Interface 只有一个输入
// CPU Control Signals
```

接下来就可以正式开始着手写两个 mem 了，计划写一个顶层调用加 data_mem 与 instr_mem。

不过在此之前，还是要搞清楚剩下的这些接口分别是些什么意思。

// Core ID, Cluster ID and boot address are considered more or less static

boot_addr_i[31:0] 启动地址