上回说到看到 riscv_core 的对外接口们，这些对外接口如下：

// 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

对这几个大类的各个接口进行分析，由于要做的是仿真，此处先优先对 input 接口进行分析。

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

//boot_addr_i 接口作进一步解释，该信号来自 PULPino 中的 SoC Controller 模块（参考图 8-3，源代码位于 https://github.com/pulp-platform/apb_pulpino ），后者内部有一个寄存器 Boot Address，该寄存器定义了系统运行的起始地址，该寄存器的值通过 boot_addr_i 接口传入 RI5CY，RI5CY 的 if_stage.sv 中的如下代码，给出第一条指令的地址。

写到这个地方，我产生了一个疑问，一个像 RI5CY 的开源项目里面应当有完备的测试模块，可以帮助节约大量时间，基于这个想法，我决定重新回到目录里面去寻找。

Tb 文件夹里面有大量的模块，决定放下现在的项目去看看这个文件夹里面的项目。

**注意“project5”和“cv32e40p-master-4”是初步 debug 好的项目**

分析下几个接口：data 接口和 instr 接口有几个共通的输入输出接口，其中有几个，名字叫：

```
output logic                        instr_req_o,
input  logic                        instr_gnt_i,
input  logic                        instr_rvalid_i,
```

和：

```
output logic        data_req_o,
input  logic        data_gnt_i,
input  logic        data_rvalid_i,
```

首先看 data_req_o，这个输出接口的意思应该是请求访问对应的数据，同样的 instr_req_o 也是一样的。Data_gnt_i 是外部数据存储区发来的，意思是我已经接受了你的访问请求。Data_rvalid_i 是表示我的数据访问完了，接下来就看你的了。

Instr 同理。

看完了以上这些，我们再看看别的几个：

```
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
```

data_we_o 表示操作类型，1 为写，0 是读。

Data_be_o 是一个 4 位信号，输出使能标志，标志着能输出整字、半字或者其它。

Data_addr_0 是数据的地址

Data_wdata_o 要写的数据

Data_rdata_i 是请求返回的数据

重新写一个太麻烦，要是能去 pulpino 找一个改改就好了。去哪找，前面已经说过了，core_region.sv。

在接口上弄了好久，最好的办法还是直接去文件夹下面搜索 rom。

第二天继续到 pulpino 项目里面去找两个 ram，昨天已经有点头绪了，找到了两个疑似模块，分别是 dp_ram 和 sp_ram。其中 dp_ram 可以完美地对应上锁需要找的 data_ram。它不仅有一大片储存模块（虽然比较少，只能存 256 个 32 位字），还可以使用输入使能，这个是非常重要的，说明要是想写一个 data_ram，就可以照着这么来写。

此外，就是寻找 instr_ram 了。

对于这个模块，首先在 core_region 里面找，根据模块头 parameter 值中的 INSTR_RAM_SIZE，找到了模块 instr_ram_wrap。之后找到的两个子模块分别是 sp_ram_wrap 和 boot_rom_wrap。其中 boot_rom_wrap 里面有我已经找不到 data 和 addr，所以就不要再找了，于是我们看向了 sp_ram_wrap。在这里我们找到了一个模块 xilinx_mem_8192x32（其中 8192 就是 32768/4，*32 很显然了），很明显，这应该是一个 ip 核。这个时候看来就只能自己写了。

下面首先试着写一个 data_ram，用 sv 语言来写。

看看首先有什么输入输出信号列在下面：

```
// 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`（output 不管）
```

写了好久都达不到要仿真的要求，这个时候，回到目录去看看，我找到了一样有很大帮助的好东西-----user_manual.doc,一个 68 页的 RI5CY 开发者文档。这个开发者文档里面有很多具体的接口和介绍，对仿真很有用处。

这个时候想到应该完备的开源项目会提供很多有用的东西，基于这个，再找找目录看看。

找到了些测试文件，debug 一番后又遇到了 mailbox 问题。

改写了下课题保存下防止丢了：

主要参考来自于 RI5CY 项目中的 user_manual.doc 文档，以及 PULPino 源码项目中的 data_sheet.pdf 文档。

目前项目的搭建已经完成，仿真模块已经导入，做出过仿真图，但在模块导入后遇到了一些问题，这些问题是可解决的！

本月的工作计划是在月底之前将完整的信号仿真图做出来，可以真正的进行测试并开

始改写。由于整个 RI5CY 的代码不仅庞大而且非常紧凑，不可能在不完全了解的情况下完成全部的代码改写，本课题的改写将试着在部分模块中进行。

改写包括：结构简单化，将其代码简化，去除冗余模块，最终的理想结果是将原来的 RI5CY 核心转写成一个仅支持 RV32I、去除冗余功能，能用来给体系结构本科生当大作业的 CPU！

主要研究学科是计算机系统设计方向，嵌入式设计的软件部分。

PULPino 是一款由瑞士苏黎世联邦理工大学的综合系统实验室和意大利博洛尼亚大学的高能效嵌入式研究组联合设计研发的 PULP 框架的简化版本。PULP 是一款多核 SoC，它的诞生是源于在某些领域，由于搜集数据能力的增强、数量的增大，导致设备将数据传送到计算平台做集中计算的代价也步步高升，对于使用电池供电的设备，这样做的消耗是十分巨大的，因此，有科学家想出直接在设备上做一个延伸，让计算直接在设备上完成，这不仅需要较强的运算能力，也需要其功耗小到可以被设备接受，PILP 因此诞生。

而 PULPino 是 PULP 的简化版本，它取消了多核心的设计，支持两种核心。本课题所采用的 PULPino 核心是 RI5CY，一款四级流水线的 32 位处理器，支持最基本的 RV32I 指令集，并包含了若干种扩展指令。本课题将着力于 PULPino 核心 RI5CY 的仿真，具体地分析并解构仿真的波形图，并在不影响其运行的情况下进行一定的改写。

Computer System Design-Simulation and Rewrite of the core of PULPino

计算机系统设计-PULPino 核心的仿真与改写

使用 tb_riscv_core.sv 没用，应该想想别的方法。

这几个输入输出信号可以直接连到 ip 核上：

```
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,
```

这三个要经过特殊的处理:

```
output logic          data_req_o,在 core_region 里面是 core_lsu_req
input    logic        data_gnt_i, 在 core_region 里面是 core_lsu_gnt
input    logic        data_rvalid_i, 在 core_region 里面是 core_lsu_rvalid
```

data_req_o 的作用是在访问中始终置为 1，一直到 data_gnt_i 置为 1 一个周期为止。如前面所说，data_req_o 在 core_region 里面和 core_lsu_req 相连，它主要是和另一个信号 is_axi_addr 决定 core_data_req 和 core_axi_req，另一个信号 is_axi_addr 是看 core_lsu_addr 的高四位的值是否为 001，否为 1，是为 0。而 core_lsu_addr 就是 data_addr_o 由于 0x10000 的值就是产生别种情况的临界地址。（可能这个信号并不是那么地重要，看看别的信号）

data_gnt_i 的作用是为 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，在 core_region 里面是 core_lsu_gnt。

太难了，想方法先全看一遍，然后直接改？

Riscv_if_stage 模块，是 RI5CY 内核里面的第一个模块，首先看输入输出接口。

首先，clk，rst_n 不用讲了。

然后看其他的，首先是

下面的子模块，一个是 riscv_prefetch_buffer 和 riscv_prefetch_L0_buffer。这两个 buffer 是用来处理由于多个处理器核共享指令缓存所产生的延迟，如果想要改写的话，由于是 pulpino，所以可以将这一部分删除。

对于 riscv_if_stage 模块，它首先继承了一个 riscv_defines.sv，这个文件里面包含了许多要执行流水线需要的数据，比如 OPCODE 系列，ALU 系列。

在 riscv_if_stage 打开后的第一块小模块，可以看到它的注释是// exception PC selection mux。这一部分的功能不怎么明朗，要说猜测的话可能是决定当程序出错的时候会产生怎么样的后果。

第二块的注解是// fetch address selection，是一个 always_comb 即组合逻辑语句它先将 fetch_addr_n 线置为 0，然后根据 pc_mux_i 线的值置为不同的值。这里置的值应该是用于在 buffer_prefetch 里面取指令的。举个例子：当 pc_mux_i 等于 PC_JUMP 的时候，fetch_addr_n 等于 jump_arget_id_i，这个值在 riscv_core 里面是由 riscv_id_stage 给出来的。很显然这就是跳转指令对应的 PC 位置。

接着下面的模块就是 prefetch_buffer 模块，这个模块首先由 RDATA_WIDTH 决定，32 位时取 riscv_prefetch_buffer，128 位取 riscv_prefetch_L0_buffer，尽管感觉由于是 pulpino，这两个模块尽量别删，因为硬件循环需要这些东西。

紧接着两个// offset FSM state 不知道是什么，但应当和分支指令有关

// take care of jumps and branches 显然是控制 jmp 指令和 ben 指令的。

之后的//hardware loops 是 RI5CY 特有的硬件循环指令的处理模块

再往后的 riscv_compressed_decoder，看代码是用来将压缩指令扩充的模块，这个模块的用处是，由于 RI5CY 还支持 16 位编码，这个模块的作用是把指令同一转换为 32 位指令的形态。

然后的 // IF-ID pipeline registers, frozen when the ID stage is stalled 是最经典的流水线间寄存器兼任阻塞功能，以前的大作业里面有过，很容易看出来。

在这个模块之前有一个// prefetch -> IF registers，这个大概就是四级流水线和五级流水线的主要不同之处了，四级流水线将五级流水线的第一级和第二集给结合起来了，结合的地方就是 riscv_if_stage 这，是连在一起的。

然后的一小片的注解是 assertions，断言？

然后看 riscv_id_stage，这个模块除了引用了上面提到的 riscv_defines 还用到了另一个 apu_core_package，它的注释上面写的是 core package of RISC-V core for shared APU，我猜测它的功能可能是和时钟同步相关，后面再看。

首先 id 模块的一大片就是解析 instr 一个 32 位指令，使用了大量 assign 语句将 instr 经过各种拆分组合变成别的微指令，这一部分大概一直到五百多行才结束。

此外，还有别的模块，紧接着的就是 hwloop 模块，即 RI5CY 特有的硬件循环模块。

注：硬件循环的功能：

这周还有其他课程的两个大实验，所以做的东西比较少，下周就没什么事情了，可以全部投入。