一些经验

在开始上手CPU之前，建议先完成以下几点：

- 配好仿真和综合所需要的环境（配好了环境就等于完成了项目的一半任务🐶）
- 搞明白架构(Pipeline/Tomasulo)的原理并对整体实现有初步构思(可以画个草图)
- 想明白Verilog语言与实际电路的关系，并理清时序逻辑和组合逻辑;

> 对于Verilog的语法，不用太在意。你要做的是按照语法规则来描述电路

环境配置

> 本人采用mac(intel chip) + VMware(Ubuntu)

仿真

- 一个好用的文本编辑器即可，最好能提供高亮、语法纠错和自动补全。

  一种可行的实现方式 : vscode + **iverilog** + 插件

  - `tabnine`：不太聪明的自动补全
  - `Verilog-HDL/SystemVerilog`：提供语法高亮和纠错
  - `waveTrace`：好看的波形(.vcd)查看器
- `riscv-toolchain`：用于将`.c`文件生成`.data`文件(仿真输入)和`.bin`文件(综合输入)

  实际上并不需要下载，因为testcase固定 => 所需要的.data 和 .bin 文件都固定

  但推荐下载，你可以自定义.c来测试你的CPU

  各系统安装教程 by Zhou Fan

综合

wsl2并不能使用，建议安装虚拟机上板，这里提供虚拟机上板一条龙服务：

首先我们要成功的在电脑上用VMware安装Ubuntu虚拟机。

安装成功后，我们需要让虚拟机能和主机文件交互，安装`vmTools`

```
$ sudo apt-get update
$ sudo apt-get install open-vm-tools-desktop
```

安装g++运行所需要的环境

```
$ sudo apt-get install build-essential
```

安装上板时与串口交流所需的包

```
$ sudo apt-get install python3-pip
$ sudo pip3 install -U catkin_tools
$ sudo apt install catkin
$ sudo apt install cmake
$ sudo pip3 install empy catkin_pkg
```

make serial库(在serial目录下)

```
$ cd serial
$ make
$ make install
```

赋予全部.sh文件读写权限(在riscv目录下)

```
$ sudo chmod +x *.sh
```

生成上板所用的控制器ctrl(在ctrl目录下)

```
$ sudo bash build.sh
```

在安装vivado之前需要**安装一个包**，否则有可能卡在安装界面，那就令人难受了

```
$ sudo apt install libtinfo5
```

接下来安装Vivado，首先下载压缩包，解压之后在`Xilinx_Vivado_SDK_2018.2_0614_1954`目录下

```
$ sudo ./xsetup
```

选择第一个Webpack版本即可。安装完毕之后，打开vivado也有一些玄机

```
$ source /opt/Xilinx/Vivado/2015.1/settings64.sh
$ mkdir ~/vivado
$ cd ~/vivado
$ vivado &
```

离上板还有最后一步，安装连接板子的驱动

```
$ cd $PATH_TO_VIVADO/2018.2/data/xicom/cable_drivers/lin64
$ sudo cp -i -r install_script /opt
$ cd /opt/install_script/install_drivers
$ sudo ./install_drivers
```

恭喜你，所有的环境都已经安装好了，有关上板的具体操作详见vivadoDemo by 学长

关于改变板子频率的方法详见ChangeFreq by 学长

关于Verilog

> verilog是一门硬件**描述**语言，意味着这门语言的目的就是为了描述物理意义上的实际电路，而不是编程语言**线性**的执行计算，所以学习Verilog的关键是理解语法与实际电路的对应关系（当然是一定抽象层级的电路）。

- 敏感信号分为电平敏感型和边沿敏感型，电平敏感型一般综合成组合电路而边沿敏感型对应时序电路
- 可综合语句：过程语句always、持续赋值assign、过程赋值=/<=、条件语句、for、`define编译指示语句

  表示这些语句可以在综合器中变成**实际**的电路；而其他语句只在仿真中起作用。

  (我认为仿真是计算机按照电路的顺序和时序模拟执行CPU当中的操作)
- 使用always块描述组合电路时，如果有信号在某些分支下未被赋值，则会引入**锁存器**（我们的项目中应避免锁存器的出现）。

  这是因为组合逻辑并**不能**保存状态，而未被赋值意味着**要保持**上一个状态。

  解决方法：将`case/if`分支语句写完整来赋予默认值，或者在分支前统一赋默认值。
- 阻塞赋值与非阻塞赋值的特点实际上是其综合出来电路的**表征**
  - 阻塞赋值：从对应寄存器采样是从**其输入端**进行的，这样读取的是新的数据
  - 非阻塞赋值：从对应寄存器的**输出端**进行采样，读取的是老的数据，而新的数据下一个周期的上升沿才更新到输出端
- `reg`变量综合时，如果**仅**用于组合逻辑(电平敏感always)，是**不会**被综合成寄存器的（这是由于在组合逻辑电路描述中，将信号定义为reg 型，只是为了满足**语法要求**）而会被综合成wire；如果有分支对reg没有赋值，则为了保有上一个时期的状态，会生成锁存器。

  如果用于时序逻辑(边沿敏感)，有概率综合成触发器，关键在于有没有**存储状态**的需求。
- 同一个时序always块中对同一个变量进行**多次**非阻塞赋值，变量的结果为最后一次非阻塞赋值的值。

  （可能是综合器自动综合成了多路选择器和电路）

  不能在多个always块中对同一个寄存器赋值，会产生数据竞争
- 时序逻辑生成触发器：低电平不能修改寄存器中的值，自然输出也不会发生变化；电平一旦升高，来自上一级的输入会被读入到寄存器，输出也发生相应的变化(通过组合逻辑抵达下一级寄存器)。由于每一级动作的同步性，而且采样时间在电平升高前一点，输入总是上一级上一时钟周期上升沿产生的输出。
- `always@(posedge clk)`中虽说是描述时序逻辑电路，但往往也有组合逻辑（比如 a <= a + 4 加法器），关键在于寄存器，来自上一级的输入通过组合逻辑产生了输出，堵在寄存器，等到电平升高才能存入寄存器。可以想像组合逻辑是奔腾的河流而寄存器是其中的大坝，电平升高时开闸使得数据能够继续流动。
- 一些我学习过程中阅读过的文章：

  何时用组合逻辑或时序逻辑、阻塞赋值&非阻塞赋值、为什么用时序电路实现CPU、阻塞和非阻塞赋值引申出的原则分析、组合逻辑和时序逻辑的综合、组合逻辑
- 学习语法网站：

  HDLBIT、菜鸟教程

可能有用?

- RV32I基础整数指令集
- 程序的终止是通过向0x30004地址写入数据实现
- io的实现通过读/写0x30000地址
  - 0x30004和0x30000的判断实际上以address[17:16] == 2'b11

    ```
    if (~q_io_en & io_en) begin //io_en: address[17:16] = 2'b11; q_io_en <= io_en
      if (io_wr) begin
    ```

    两个信号才能进入if当中，其中一个信号滞后另外一个信号一周期，会导致奇怪问题（不能先传地址；必须有间隔）
  - `io_full`信号的更新滞后申请写0x30000两周期，所以每次写0x30000后要等待两个周期再尝试下一次写
- RAM的实现：无论读写都不是纯组合逻辑

  读也要滞后两个周期才会有数据回传(来自于读地址的时序逻辑)
- 以及 sleep() 和 inl()函数在仿真时不能跑，直接注释掉或者换成输入的值即可。