UCAS undergraduate opening report

Yingkun Zhou, 2015K8009929023

0.Opening

我一直有个比喻在计算机领域，体系结构者扮演的是上帝的角色。上帝造万物，制规律，但却察觉不到。体系结构不就是如此吗，人们只关注其上跑的应用如否满足自己的需求，是否流畅 (哪怕这个体系结构的 ISA 再烂，比如 Intel)，如果不是计算机系的人，根本不会关注其背后的规律，何种指令以何种方式被执行。而操作体统犹如国家

1.Background

体系结构领域出现了一股不可忽视的力量，其影响力将不啻 OS 领域的 Linux，这就是 risc-v。而我选择 risc-v 作为本科毕业设计的大的框架背景也是处于如下几个方面的考虑。

- 在今年的龙芯杯中，自己设计的 CPU 以及在上面做的 software stack 的工作虽然侥幸获得了第一，但是

2.language and logic

首先语言的更高层次化是为了代码的简洁与复用，换言之就是由编译器来做琐碎的事情，从而解放人去更多的思考逻辑的事。但是语言的更高层次化也不应该提高代码的逻辑复杂密度。不得不承认，从 C 到 C++ 以及 Java，代码的逻辑复杂密度有所提高，C 是简单的二层逻辑–global & function local; 面向对象则是各种类的继承，抽象，方法的多态辅以各种一时难以理解的 λ-函数，iteration 迭代，代码结构难以 trace。如果代码框架是自己写的，那自然非常 happy，因为编译器帮你做了很多琐碎的事情，但是旁人如果想修改一开始就非常头疼 (举个最简单的例子，C 代码中如果你想标准输出，想都不用想直接调用 printf，但是 Java 中呢，你就必须得知道 print 函数是在哪个 class 中，那个 class 又是在哪个 package 中，像我初学 Java 的时候真的是每一个想调用 print 的时候就必定会查 google；还有一个毛病就是 C 可以 GDB，当然 C++ 也可以 GDB，但调起来就有些头疼了，有时候你根本不知道代码会什么会跳到那里 (就是前面说的难以 trace)，Java 的话貌似都没有 GDB 这么一说，每次我都用 print 大法)。

但是言归正传，面向对象语言所付出的这些代价都是值得的，尤其是在硬件逻辑的编写领域。为什么这么说呢？现在任给你一张 CPU 的架构图，一定是一些方框的逻辑组合，这些方框又是什

么呢？就是模块，既然是模块，那么目前最好的刻画方式就是面向对象的语言。我一直很好奇为什么我们能够忍受这么多年的 Verilog，难道高层的语言解决不了的电路的逻辑吗？一开始我是这么认为的，因为 sequential circuit 的刻画就像是盘亘在高级语言上的一座大山, 如果仔细一想，哪有语言想要去刻画 reg。但是直到接触了 chisel，才惊呼高实在是高。

我们可以充分利用类的继承来简化繁琐的模块之间的接口与连线