LC-3 实验指导手册

实验目录

实验一：搭建 Chisel 开发环境

1. 实验目的：

1.1. 了解芯片设计开发流程，建立基础概念

1.2. 掌握 Chisel 开发环境的搭建

1.3. 根据实验指导编译与运行 3-8 译码器项目

2. 实验内容：

2.1. 学习关于开发流程的背景知识

2.2. 学习安装及使用 Vmware 虚拟机软件

2.3. 学习安装 Chisel 开发环境

2.4. 编译运行 3-8 译码器项目

3. 实验步骤：

3.1. 芯片前端设计开发背景知识

在一个完整的芯片从确定需求到制作完成，其中包含了许多的流程，可以大致分为设计和制造两大部分，其中设计通常是由开发芯片的公司完成，确定芯片的各种性能参数和规格，设计完成后，将芯片的详细设计图纸送往芯片代工厂，例如中芯国际、台积电等，由他们完成芯片制造这部分的工作。芯片设计可以理解成一栋大楼的设计师，设计师绘制大楼的建筑图纸，确保这栋大楼结构的稳固和功能的完备。而芯片制造则是负责真正一砖一瓦的将这栋大楼建造完成的工人。

而芯片的设计部分又可以细分为前端设计（也称逻辑设计）和后端设计（也称物理设计）。前端设计主要关注的是芯片如何实现需要的功能，怎样达到性能要求，而后端设计主要关注的是如何保证前端设计出来的电路在现实中能够制造出来并正常运行。在本实验课中，主要学习了解的是芯片前端设计的流程，而后端设计流程会在另一套实验课中介绍。

芯片前端设计的主要流程如图 1.1 所示，接下来详细介绍各个流程的含义。

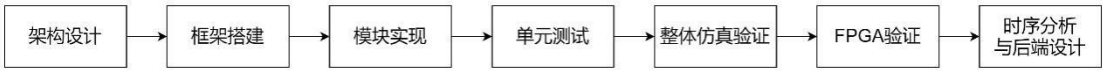

图 1.1 芯片前端设计流程

架构设计：和软件设计一样，在决定设计制作一款芯片之前，需要进行需求分析，确定芯片的功能、性能、可行性，然后依此设计相关的架构，按功能划分模块，确定各部件的规格，形成一份设计文档，后续开发以此文档作为参考。

框架搭建：设计文档完成后，需要尽快开始搭建整体项目的框架，搭建好开发环境，配置好项目构建工具，为后续模块的开发及验证做好铺垫。

模块实现：按照设计文档，将整体功能划分为多个模块，再将多个模块划分为更细的各个子模块，然后逐一实现，在模块实现的过程中，使用 HDL (hardware description language, 硬件描述语言) 来描述电路的实现，通过 HDL，能够精确地描述电路设计中每一个寄存器，每一个逻辑门，每一根线，而通过 HDL 描述的电路代码称为 RTL (Register Transfer Level, 寄存器传输级) 代码，这也是前端设计最终交付给后端做物理设计的目标代码。HDL 有很多种，例如 Verilog、System Verilog、VHDL 等，在本实验中，使用 Chisel 作为主要开发语言，在接下来的一系列实验中，会逐步的介绍 Chisel 的相关基础知识。

单元测试：模块实现完成后，需要对其进行测试，因此需要针对模块编写对应的单元测试，单元测试主要测试模块的功能正确性，应该随模块一起开发。之所以需要单元测试，是因为单元测试的目标模块相比于整个系统更加简单，在模块实现错误时可以快速定位错误，而且单元测试的速度极快，可以通过快速地迭代，即发现问题-修改代码-再次测试不断循环，直到模块通过测试。确保模块功能的正确性，提升开发效率。单元测试的设计也很关键，决定了对模块的测试是否充分，单元测试越全面，后续这个模块出错的概率越小。

整体仿真验证：在所有模块的单元测试都通过后，需要将所有的模块整合在一起，整个系统进行仿真测试，之所以叫做仿真 (Simulation) 测试，是因为这个阶段不可能将设计代码用真实的物理器件做出来测试，需要使用专门的仿真软件，来模拟设计的电路的行为，来验证功能的正确性，因此叫做仿真测试。这一步目的在于验证整体系统的功能，可以发现一些模块间交互的错误，或者之前单元测试没有覆盖到的情况。并且此时可以通过运行一些标准测试程序，来验证整体设计的性能。

FPGA 验证：在仿真验证通过之后，可以通过将设计在 FPGA 上实现来近似的验证，FPGA (Field Programmable Gate Array, 可编程逻辑门阵列)是一种半定制电路，可以简单的理解成一块可以随意改写内部电路的芯片。在 FPGA 上验证可以发现一些仿真时容易忽略的细节，比如通信协议，复位信号的实现。由于 FPGA 上调试相比于单元测试和仿真时更加复杂，因此尽量在整体仿真验证时确保所有功能正常。

时序分析与后端设计：在 FPGA 验证通过后，就可以进行时序分析，以及后端的物理设计。时序分析主要是检查设计的电路是否能够满足设计文档中一些频率等指标的要求，后端设计主要是将 RTL 代码转换为可以用于代工厂制造芯片的文件。

以上就是前端设计的开发流程，在接下来的实验中会逐一介绍时序分析之前的每个流程。介绍相关的各种开发工具和基础知识。而本次实验中，会带领大家实现一个简单的小模块，并在电脑上仿真运行它的单元测试。本次实验首先带大家熟悉流程，编译和仿真过程中，每一步的原理会在实验二中详细介绍。

3.2. Vmware 虚拟机软件的安装及使用

在项目开发时，通常会使用 Linux 操作系统做开发，因为 Linux 系统相比于 Windows，在长时间运行程序时会更加稳定，并且具有强大的终端操作能力，且是目前行业内普遍使用的操作系统，因此建议在 Linux 操作系统下搭建开发环境。考虑到大部分同学的电脑都是 Windows 系统，因此建议在 Windows 上安装虚拟机软件 Vmware，这样做的好处是首先不用重装系统，在不使用虚拟机时电脑可以当作普通的 Windows 系统使用。其次使用虚拟机可以更好的对整个系统进行备份，在出现错误操作时可以快速恢复。同时搭建开发环境费时费力，可以直接导入实验材料中搭建好环境的系统镜像。Linux 系统的版本我们选择 Ubuntu20，具体 Vmware 的安装和 Linux 系统的安装，如何生成备份（快照），这部分作为附件，在《虚拟机安装与使用》文档中查看。在安装 Vmware 和导入虚拟机时，请保证硬盘中还有足够的空间。

3.3. Chisel 开发环境的搭建

3.3.1. Chisel 介绍

Chisel (Constructing Hardware In a Scala Embedded Language)是美国加州大学伯克利分校开发的一种开源硬件构造语言，正如它的全称一样，是一种构建在 Scala 语言之上的领域专用语言（Domain Specific Language, DSL），而 Scala 是一种高级的软件编程语言，运行在 Java 虚拟机上。Chisel 相比于传统的硬件开发语言，有更多高级语言的功能。在编译时，会将 Chisel 代码编译生成 verilog 文件，然后再使用 verilog 文件仿真，其中 verilog 是一种传统的硬件设计语言，与 Chisel 的关系类似于汇编语言和 C 语言的关系。但是在开发过程中，可以只使用 Chisel 语言设计电路，而不用学习 verilog 的语法。Verilog 文件只是一个中间文件，不用过度关注 verilog 文件的具体内容。接下来在虚拟机中尝试搭建 Chisel 的开发环境。*（由于安装过程花费的时间比较长，可以选择直接导入搭建好环境的虚拟机镜像，但请仍旧详细阅读一遍实验指导）*

再整理一遍各种语言之间的关系，**Scala** 是一种软件开发语言，类似 C++和 Java，而 **Chisel** 是一种硬件开发语言，是集成在 Scala 语言中的，而 **verilog** 是传统的一种硬件开发语言，Chisel 语言在编译时，会先编译成 verilog 语言，再进行之后的操作。Chisel 和 verilog 类似于 C 和汇编的关系，Chisel 会更高级，verilog 更底层。

3.3.2. Java 介绍（如果使用虚拟机，则忽略此步骤）

如上文所示，Chisel 需要在 Java 虚拟机上才能编译运行，Java 是一种能够跨越多平台的、具备高度可移植性的面向对象的编程语言，也是目前最先进、特征最丰富、功能最强大的计算机语言。因此搭建 Chisel 编译环境首先需要安装 Java 的开发包 JDK (Java Development Kit)，其中包含了 Java 虚拟机在内的完整 Java 开发工具。

JDK 直接使用 apt 工具就能够安装，在终端中输入以下命令：

```
sudo apt install -y openjdk-13-jdk
```

输入当前账号的密码之后等待命令执行完成。注意输入密码时密码不会出现在屏幕上，这是正常现象，输入完密码按下回车命令即会继续执行。在命令执行完成后，通过输入以下命令，查看刚才安装的 JDK 的版本号，出现如图 1.2 所示的输出时则代表 JDK 安装成功。

```
java --version
```

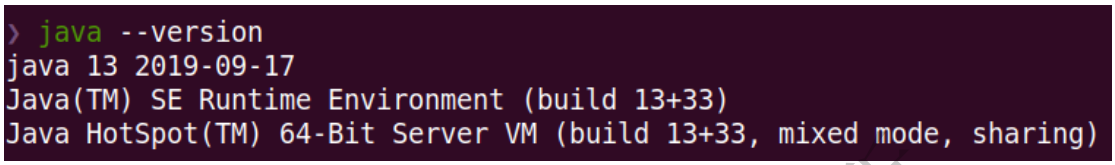

图 1.2 查看 Java 版本号

3.3.3. mill 介绍

在 JDK 安装完成之后安装 mill。Mill 是一个项目构建工具，主要用于将 Chisel 编译成 verilog 文件。通过配置相关的文件，可以指定要使用的 Chisel 和 Scala 的版本。

mill 无法简单的使用 apt 工具安装，需要手动地将其安装文件下载下来然后运行，首先需要在 Linux 上安装 curl 程序，curl 是 Linux 常用的一个下载工具，用于从指定的网站下载文件，curl 可以直接使用 apt 安装，运行如下命令：

```
sudo apt install -y curl
```

curl 安装完成后，在终端中输入如下命令进行安装：

```
sudo sh -c "curl -L https://github.com/com-lihaoyi/mill/releases/download/0.10.2/0.10.2 > /usr/local/bin/mill && chmod +x /usr/local/bin/mill"
```

在安装成功后，输入以下命令，会开始从外网下载相关文件，速度较慢，也有可能下载失败，需要多尝试几次，直到最后能够看到 mill 的版本号，则代表安装完成。

```
mill --version
```

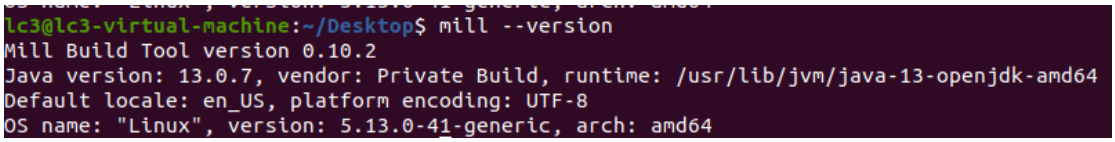

图 1.3 查看 mill 版本号

3.3.4. Verilator 介绍（如果使用虚拟机，则忽略此步骤）

Verilator 是一种开源免费的 verilog 仿真器，可以通过软件模拟的方式仿真运行硬件电路，主要工作原理是读取 RTL 代码，将其转换成用于仿真的 C++项目，再通过编译运行 C++项目，就可以在电脑上对硬件电路进行仿真。

Verilator 的安装直接使用 apt 工具即可，运行以下命令

```
sudo apt install -y verilator
```

安装完成后运行以下代码，看到输出版本号后代表安装成功

```
verilator --version
```

mill 和 Verilator 安装完成后，Chisel 的开发环境就搭建好了，在下一小节将给出一个简单电路的实现，并详细介绍如何对其进行仿真和测试。

3.4. 3-8 译码器的编译和运行

本节主要介绍一个简单的模块：3-8 译码器。译码器（Decoder）顾名思义，就是将带有某种规则的编码，翻译成另外一种规则的信号的电子器件。3-8 译码器则是将一个 3 位的二进制数 n 转换为 8 个信号，其中第 n 个信号为高电平，用 1 表示，其余的信号为低电平，用 0 表示。如表 1.1 是 3-8 译码器的真值表。

表 1.1 3-8 译码器真值表

in	in[2]	in[1]	in[0]	out[7]	out[6]	out[5]	out[4]	out[3]	out[2]	out[1]	out[0]
0	0	0	0	0	0	0	0	0	0	0	**1**
1	0	0	1	0	0	0	0	0	0	**1**	0
2	0	1	0	0	0	0	0	0	**1**	0	0
3	0	1	1	0	0	0	0	**1**	0	0	0
4	1	0	0	0	0	0	**1**	0	0	0	0
5	1	0	1	0	0	**1**	0	0	0	0	0
6	1	1	0	0	**1**	0	0	0	0	0	0
7	1	1	1	**1**	0	0	0	0	0	0	0

举例说明，如图 1.4 所示，给 3-8 译码器输入二进制的数 110，换算成十进制为 6，对应的，在译码器的 8 位输出中，第 6 位为高电平（位数从 0 开始算），而其他的位为低电平。

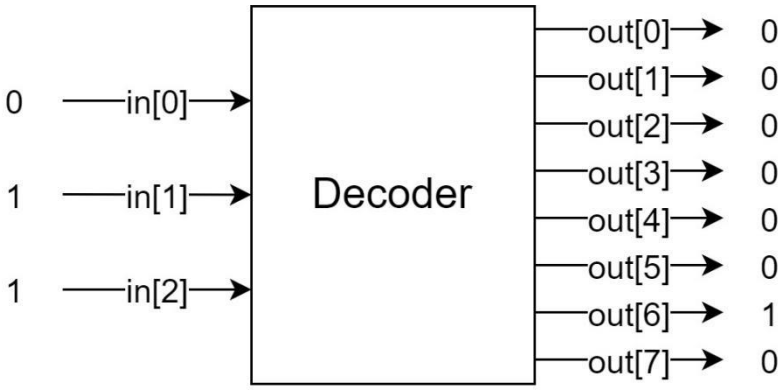

图 1.4 译码器示例

各式各样的译码器在电路设计中应用非常广泛，可以用于将复杂的编码翻译成易于电路使用的更加直观的信号。

接下来将实验材料中的 decoder 项目文件夹拷贝到自己虚拟机的目录下，如果安装了虚拟机的增强功能，可以直接从电脑上拖动文件夹到虚拟机内部的目录。接下来打开一个终端，进入到 decoder 目录中，通过输入 cd decoder 命令可以切换 Linux 的工作目录，通过 cd ..可以回到上一级目录。进入到 decoder 目录后，输入 ls 命令，可以查看该目录的结构，目录中的文件应该如图 1.5 所示。请尝试通过使用 cd 和 ls 命令切换目录，确认目录中所有的文件都存在。

注意：如果使用的是实验提供的虚拟机，那么实验材料代码放在 ~/Frontend 中

```
lc3@lc3-virtual-machine:~/Desktop$ cd
lc3@lc3-virtual-machine:~$ cd Frontend/
lc3@lc3-virtual-machine:~/Frontend$ pwd
/home/lc3/Frontend
lc3@lc3-virtual-machine:~/Frontend$ ll
total 20
drwxrwxr-x  5 lc3 lc3 4096 5月  15  2022 ./
drwxr-xr-x 33 lc3 lc3 4096 10月 10 21:46 ../
drwxrwxr-x  7 lc3 lc3 4096 5月  15  2022 chisel_lc3/
drwxrwxr-x  3 lc3 lc3 4096 5月  15  2022 decoder/
drwxrwxr-x  5 lc3 lc3 4096 5月  15  2022 detection/
```

```
decoder
├── build.sc
├── Makefile
├── sim_main.cpp
└── src
    └── main
        └── scala
            └── decoder.scala
```

图 1.5 decoder 目录结构

接下来在 decoder 目录下，运行以下命令：

```
make emu
```

然后等待命令执行完成，第一次编译需要的时间可能比较长，如果出现了访问 https://repo1.maven.org 网站的错误，多尝试运行几次命令。期间可能会看到类似图 1.6 的输出。

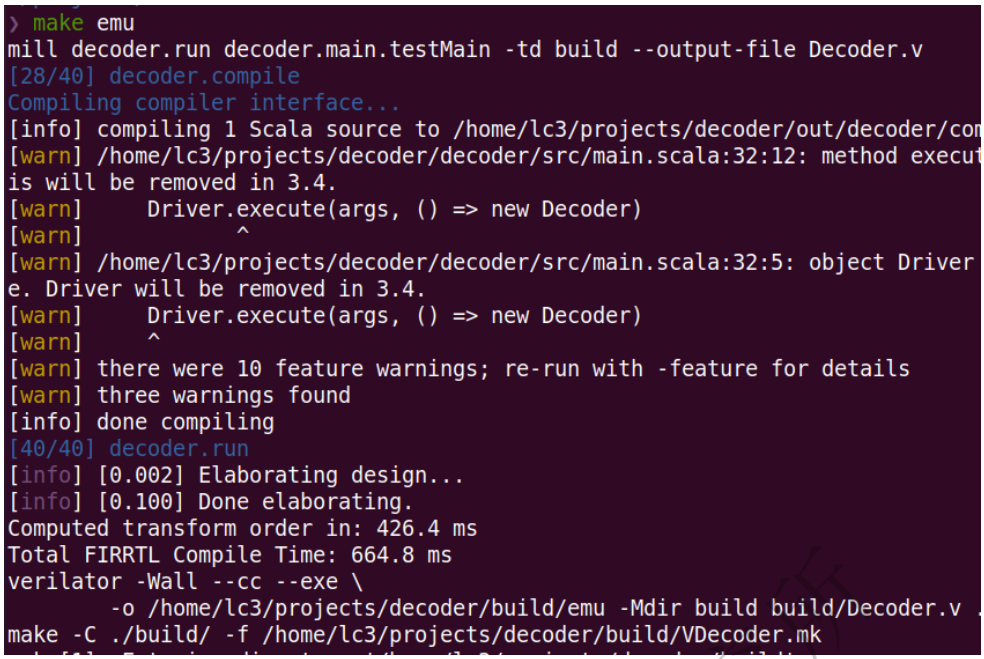

图 1.6 编译 decoder 项目终端的输出

命令执行完成后，再次在 decoder 目录下执行 ls 命令，可以看到多出了两个目录，一个名为 build，一个名为 out，这两个都是编译自动生成的目录，其中有许多的文件。接着在 decoder 目录下运行以下命令：

```
./build/emu
```

执行这个命令后就会开始对整个 3-8 译码器项目进行仿真运行，如果运行正确，可以看到如图 1.7 所示的输出结果，这可以看作是对 3-8 译码器的一个单元测试。可以对照自己的运行结果，应该与图 1.7 相同。

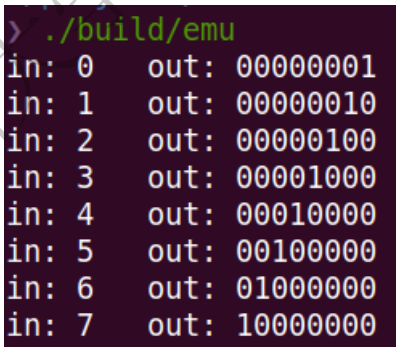

图 1.7 3-8 译码器仿真运行结果

至此已经完成了一个从代码编译到仿真运行的简单流程，也许你对每一步做了什么完全没有了解，但是至少你已经走通了这一流程，接下来的实验会介绍在这个过程中每一步的作用，相关的代码的含义。本次实验最重要的是了解芯片设计的流程，以及本实验所教授的知识在整个芯片设计流程中的哪一部分。

实验一-任务一：根据实验指导，完成开发环境的搭建，或导入虚拟机镜像（如果使用虚拟机，忽略这个任务）

实验一--任务二：根据实验指导，完成 3-8 译码器的仿真流程

3.5.