简介

Verilog HDL（简称 Verilog）是一种**硬件描述语言**，用于数字电路的系统设计。

学习目标

1. 熟悉 Verilog 的基础语法与语言特性，完成 Verilog 小作业。
2. 掌握 CPU 基础逻辑部件的架构设计。
3. 了解 CPU 的时序逻辑与组合逻辑的区别。
4. 了解 CPU 的频率、周期、时延等概念之间的区别与联系。

作业任务

- `task1`：了解 `wire` 与阻塞赋值的工作原理。
- `task2`：了解 `reg` 与非阻塞赋值的工作原理。
- `task3`：掌握 Verilog 基础语法，完成一个简单的奇偶校验，提交至 `1618. 奇偶校验` 。
- `task4`：掌握组合逻辑，完成一个加法器，提交至 `1250. 加法器` 。
- `task5`：掌握时序逻辑，了解状态机，完成给定状态转移图对应的状态机，提交至 `1616. 状态机 1` 。
- `task6`：学会使用状态机解决实际问题，完成能进行简单字符串匹配的状态机，提交至 `1617. 状态机 2` 。

环境配置

1. 推荐使用 icarus-verilog (iverilog) 作为仿真工具。
    - Linux: `sudo apt install iverilog`
    - Mac OS: `brew install icarus-verilog`
    - Windows: https://bleyer.org/icarus/ **安装时选中 `Add executable folder to the user PATH`**
2. 推荐使用 vscode 作为开发工具，安装相关插件有助于提升开发效率。
3. **请勿使用 clion 作为 verilog 开发工具**。
4. 如果有需要，可以使用 gtkwave 作为波形图展示工具进行 debug。

编译运行

可以编译运行下发的样例代码 `example_tb.v` 进行测试。

```
iverilog -o example example_tb.v
./example
```

Verilog 运行的是名为 `top_module` 的模块（可类比为C++中的 `main` 函数），在其中可以进行对其他模块的调用。

学习资料

1. https://hdlbits.01xz.net/wiki/Main_Page
2. https://www.runoob.com/w3cnote/verilog-tutorial.html

Tips

编写 Verilog 时要转变思维，**不能照搬之前编写程序的思路，而是要认识到是在编写硬件电路**，想象实际的真实硬件的架构有助于编写出正确的 Verilog 代码。

wire

1. `wire` 可简单地理解为电线，连接的过程中可以夹杂逻辑门（对数据进行运算）。
2. 因此 `wire` 只能用 `assign` 赋值，并且只能被赋值一次（因为电线不可能随意变动）。
3. 因此对 `wire` 的声明与赋值无关乎被写在 `module` 中的什么位置（因为是连接好的电线）。

reg

1. `reg` 具有储存数据的功能，不同于 `wire` 。
2. 在 `@always(*)` 中只使用阻塞赋值 `=` 给 reg 赋值，这等价于 wire 用 assign 赋值。
3. 在 `@always(posedge clk)` 中只使用非阻塞赋值 `<=` 给 reg 赋值，所有赋值操作是并行的，等价于只会使用上一周期的数据进行赋值。

综合 / 组合与时序

1. 综合：将使用硬件描述语言描述的电路转换成实际的真实电路的过程。
2. 组合逻辑综合出的电路是按代码描述的逻辑顺序运行的。
3. 时序逻辑综合出的电路是并行的，即只会使用上个周期的信息来得到输出，在代码中描述的顺序不应影响输出结果。因此一个周期内，一个寄存器应只允许被赋值一次，否则可能会出现 `multiple driver` 问题。
4. 在组合逻辑中不可以进行自赋值，否则会导致综合出 `latch` （锁存器），将会极大消耗资源甚至无法运行；因此需要自赋值时，必须使用时序逻辑。
5. 不应该使用 `initial` （下发的测试代码中已有的除外），这是无法综合的语句。
6. 只有循环次数确定（可手动展开）的循环语句可综合。