计算机组成原理实验

Lab 0 manual

Verilog 编程工具与

Vivado 模块化仿真调试

Made by TA

2023 年 3 月 20 日

目录

1 前言 **3**

2 主要内容 **3**

3 使用 Vscode 编写 Verilog 代码（可选，十分推荐!） **4**

3.1 配置 Vscode 相关插件，以及将 Vscode 作为 Vivado 的默认编辑器 4

3.2 配置实时代码语法检查 6

3.2.1 方法一：基于 Vivado 的 xvlog 工具 6

3.2.2 方法二：不依赖 Vivado 的语法检查工具 8

4 模块化设计与仿真调试（必读内容） **10**

4.1 使用 Vivado 进行模块化仿真(必读内容) 11

4.2 不依赖于Vivado的模块化仿真 14

5 实验题 **15**

6 FAQ, 配置有问题先看这里 **18**

1. 前言

在去年的计组实验中，很多同学由于不熟悉 Verilog 的编程规范，在编写代码时会产生大量简单的语法错误；或因为不了解 Verilog 的某种写法是否合法，根据直觉进行编写，直到编译时才发现硬件结构上的问题，浪费了大量时间。实际上，我们平时使用的 Vscode 配合插件可以提供包括实时语法检查、代码高亮、模板补全、自动代码生成、自动生成 testbench 等极为强大的功能。

本学期的《计算机组成原理》课程实验需要大家自己编写可运行的 CPU。一个功能完整的 CPU 涉及到的逻辑电路较为复杂，因此模块化设计与调试的思想在 CPU 设计实验中十分重要。仿真可以允许我们查看组成整个电路的各个模块的输入输出信号，而非仅仅 topmodule 的信号，从而便于我们发现到底是哪个模块的信号与预期不符。

在正式实验开始之前，我们为大家提供了本次的预热实验。本文档将重点介绍 Verilog 的编程工具，以及模块化仿真调试方法，希望能够为大家后续的实验提供有效的帮助。

> Tips：如果现在你仍然对 Verilog 语言基础的语法和单模块编程过程不熟悉，助教强烈推荐大家在 https://hdlbits.01xz.net/wiki/Main_Page 上完成相关的练习。我们的 Verilog OJ 平台上的题目大部分来源于这里，而此网站在练习题中配有更加丰富的题目讲解。

2. 主要内容

本次实验文档的内容如下：

1. 在 Vscode 中编写 Verilog 代码，并将其设置为 Vivado 的默认编辑器；
2. 学习 Verilog 模块化设计与调试方法；
3. 一道你需要完成的实验题目。

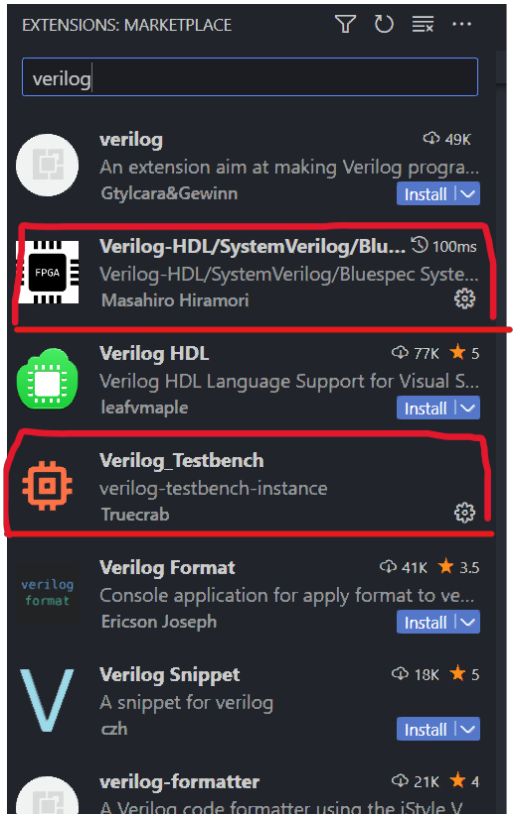
3. 使用 Vscode 编写 Verilog 代码（可选，十分推荐!）

Vscode 是一款强大的代码编辑器。在原版的 Vivado 上，编写 Verilog 程序相对而言受限较多，且没有其他有利于代码编写的拓展功能（什么年代了还在用 Dev C++.jpg）。使用 Vscode 编写 Verilog 程序的好处在于可以使用插件进行代码高亮、语法检查、自动补全等功能，也方便自建文件夹管理本课程实验的所有项目，从而大大提高了编写代码的效率。为实现上述功能，我们需要安装一些相关的 Verilog 插件，同时配置 Vscode 和 Vivado。具体可以有两种方法：

3.1 配置 Vscode 相关插件，以及将 Vscode 作为 Vivado 的默认编辑器

从零开始的步骤如下：

1. 安装 Vscode(https://code.visualstudio.com/)
2. 在插件市场搜索 verilog，安装 Verilog-HDL/SystemVerilog/... (by Masahiro Hiramori)和 Verillg Testbench(by Truecrab) 插件, 也就是下图中红线里的 Verilog 插件。



3. 配置 Vivado

   在 Vivado 中打开 Tools->Settings->Text Editor ，将其中的 Current editor 改为 Custom Editor，随后点击后面的 ... 按钮，在弹出的界面中填入以下内容：

   到 Microsoft Vscode 的绝对路径/Microsoft VS Code/Code.exe -g [file name]:[line number]

   点击 ok 后即可完成配置，此时 Vscode 成为了 Vivado 的默认编辑器。

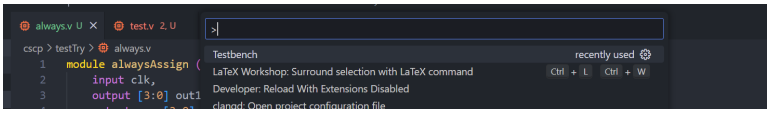
相关插件的使用介绍：

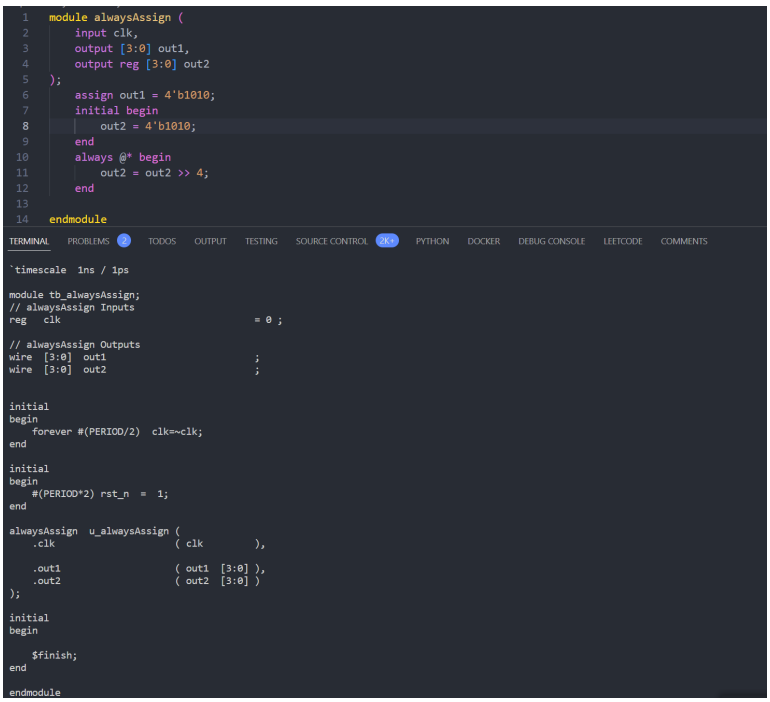
- Verilog-HDL/SystemVerilog/..., 提供语法高亮，代码补全等功能

- Verilog Testbench 可以根据当前文件内容自动生成 testbench 文件

  使用方法：在 Verilog 文件中按下 Ctrl+Shift+P 调出命令面板，输入 testbench，即可自动生成 testbench 文件。

> Tips：该插件是基于 Python 的，你可能会被提示缺少一些 Python 的库，只需要根据其提示的库名 pip install 库名即可。（当然如果你的电脑没有安装 Python 的话，需要先安装 Python）





```
module alwaysAssign (
    input clk,
    output [3:0] out1,
    output reg [3:0] out2
);
    assign out1 = 4'b1010;
    initial begin
        out2 = 4'b1010;
    end
    always @* begin
        out2 = out2 >> 4;
    end

endmodule
```

```
TERMINAL  PROBLEMS 2  TODOS  OUTPUT  TESTING  SOURCE CONTROL 2K+  PYTHON  DOCKER  DEBUG CONSOLE  LEETCODE  COMMENTS

`timescale  1ns / 1ps

module tb_alwaysAssign;
// alwaysAssign Inputs
reg   clk                                  = 0 ;

// alwaysAssign Outputs
wire  [3:0]  out1                          ;
wire  [3:0]  out2                          ;


initial
begin
    forever #(PERIOD/2)  clk=~clk;
end

initial
begin
    #(PERIOD*2) rst_n  =  1;
end

alwaysAssign  u_alwaysAssign (
    .clk                    ( clk        ),

    .out1                   ( out1  [3:0] ),
    .out2                   ( out2  [3:0] )
);

initial
begin

    $finish;
end

endmodule
```

你可根据其生成的 testbench 文件自行调整，得到符合自己预期的测试文件。

3.2 配置实时代码语法检查

3.2.1 方法一：基于 Vivado 的 xvlog 工具

> 提示：本方法需要安装 Vivado，如果没有安装 Vivado，请参看方法二。

> Vlab 平台上的虚拟机已经预装了 Vivado，如果不想在自己的物理机上进行实验，可以选择使用 Vlab 虚拟机。具体的配置过程也可以参考下面的操作。

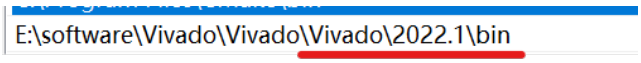
> Vivado 的安装可以参照 https://vlab.ustc.edu.cn/guide/doc_vivado.html ，其中的最小安装方法可以尽可能减少硬盘内存占用需求。

1. 配置系统环境变量(提供 Windows 和 Vlab 两种环境的方法)

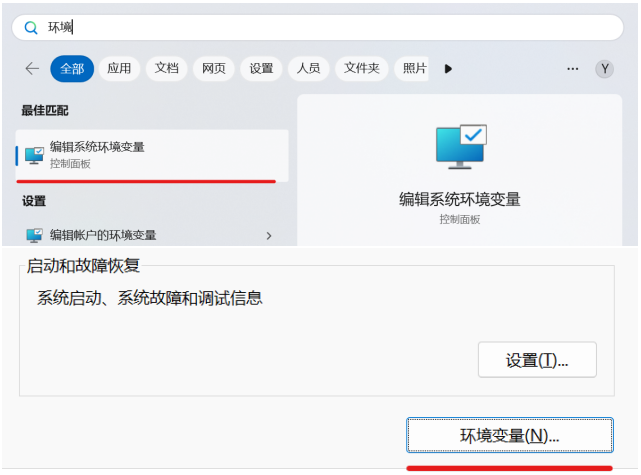
   在 Windows 环境下，你需要配置 Vivado 的 bin 目录到系统环境变量中，具体步骤为：

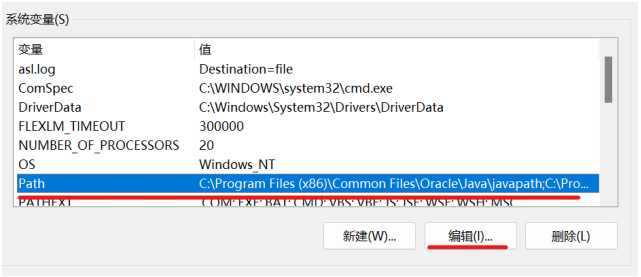
   - 找到 Vivado 的 bin 目录

     这个目录一般位于 .../Vivado/版本号/bin。找到该目录后，请复制该目录的路径（注意这一步需要复制绝对路径）



- 找到系统环境变量,步骤如下：

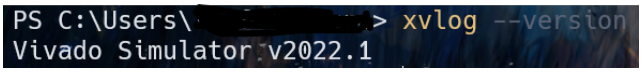




- 添加系统环境变量

  在上一步中的界面点击新建，将所复制的路径粘贴到变量值中，点击确定后即可完成环境变量的配置。

- 验证系统环境变量是否配置成功在 cmd 中输入 xvlog - -version。如果出现如下类似内容则说明配置成功, 如果失败则说明系统变量可能未应用，重启电脑即可。



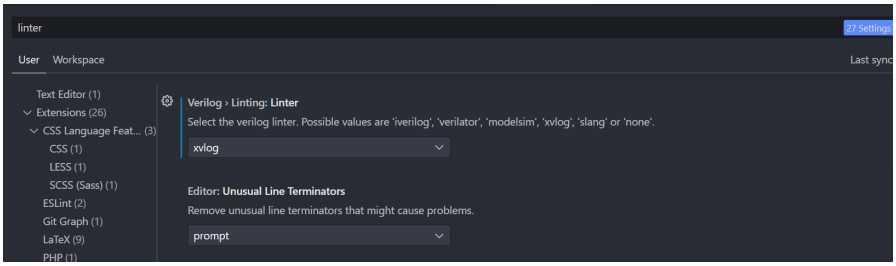
在 Vlab 下，你同样需要将 Vivado 的 bin 目录添加到 PATH 环境变量中。具体步骤如下（其他 Linux 环境类似操作）：

- 在终端中输入 vim ~/.bashrc；
- 在底部加入 export PATH="$PATH:/opt/vlab/vivado/Xilinx/Vivado/2019.1/bin/"，随后保存并退出。保存并退出 Vim 的方式为：键盘输入：+ wq（加号不用输入）。

- 在终端输入 source ~/.bashrc 使环境变量生效；
- 在终端输入 xvlog - -version。如果出现版本号则表明配置成功。

2. 配置 Vscode

   在 Vscode 中打开设置，搜索 linter 并将其设置为 xvlog。（你可能需要重启 Vscode 使其生效）



3.2.2 方法二：不依赖 Vivado 的语法检查工具

> 提示：本方法不依赖 Vivado，主要提供给使用 MacOS 的同学使用, Windows 环境下也可以使用。

1. 安装 iverilog

   在 MacOS 上，可以直接使用 homebrew 安装。如果没有安装 homebrew ，可使用科大镜像源安装 https://mirrors.ustc.edu.cn/help/brew.git.html#homebrew-linuxbrew（比官方源要快）。

   安装好 homebrew 后，在终端中输入 brew install icarus-verilog 即可完成安装。

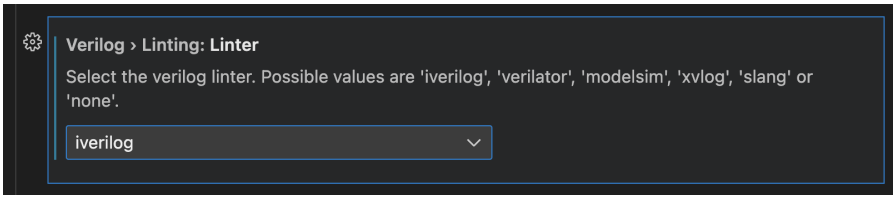
   linux 也可通过 apt 来安装；当然，你也可以在 github 上找到源码并自己编译安装。

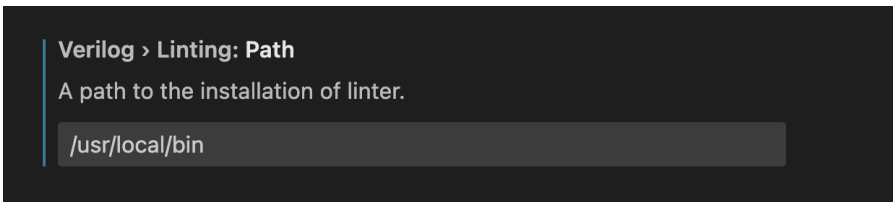
   Windows 可从 http://bleyer.org/icarus/ 获取安装程序。安装时需记住安装位置。

2. 配置 Vscode

   在 Vscode 中打开设置，找到 Verilog configuration ，将其中的 Verilog > Linting : Linter 设置为 iverilog ，并将 Verilog > Linting : Path 设置为 iverilog 可执行文件的路径。

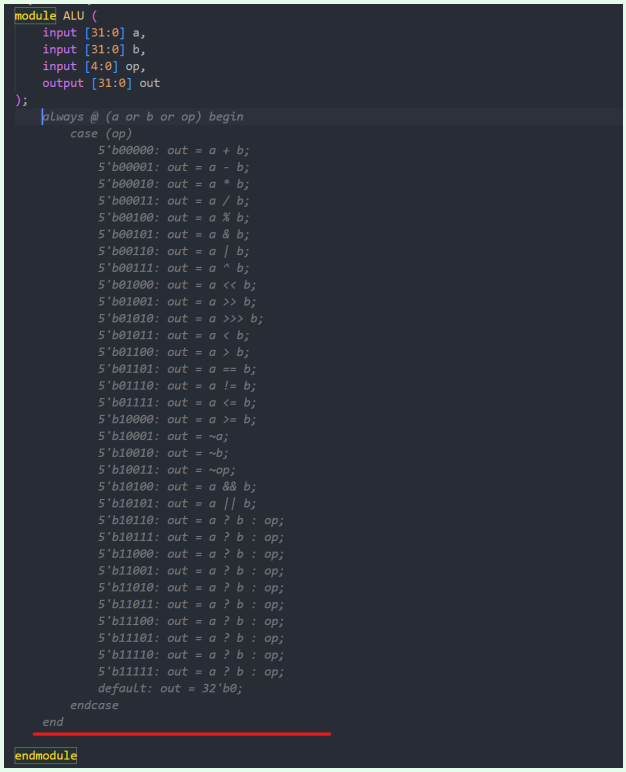
   在助教的一台电脑上的示例如下：





在 MacOS 和 Linux 上可以使用 which iverilog 命令来快速定位 iverilog 的路径。使用 Windows 时，则使用安装时记住的路径。

提示：如果大家有兴趣的话可以探索使用 tabnine，github copilot（基于 gpt-3 学生免费）等插件，它们可以提供强大的自动补全和代码生成能力。不过请不要用其不加思考地直接生成实验所需要的代码。



```
module ALU (
    input [31:0] a,
    input [31:0] b,
    input [4:0] op,
    output [31:0] out
);
    always @ (a or b or op) begin
        case (op)
            5'b00000: out = a + b;
            5'b00001: out = a - b;
            5'b00010: out = a * b;
            5'b00011: out = a / b;
            5'b00100: out = a % b;
            5'b00101: out = a & b;
            5'b00110: out = a | b;
            5'b00111: out = a ^ b;
            5'b01000: out = a << b;
            5'b01001: out = a >> b;
            5'b01010: out = a >>> b;
            5'b01011: out = a < b;
            5'b01100: out = a > b;
            5'b01101: out = a == b;
            5'b01110: out = a != b;
            5'b01111: out = a <= b;
            5'b10000: out = a >= b;
            5'b10001: out = ~a;
            5'b10010: out = ~b;
            5'b10011: out = ~op;
            5'b10100: out = a && b;
            5'b10101: out = a || b;
            5'b10110: out = a ? b : op;
            5'b10111: out = a ? b : op;
            5'b11000: out = a ? b : op;
            5'b11001: out = a ? b : op;
            5'b11010: out = a ? b : op;
            5'b11011: out = a ? b : op;
            5'b11100: out = a ? b : op;
            5'b11101: out = a ? b : op;
            5'b11110: out = a ? b : op;
            5'b11111: out = a ? b : op;
            default: out = 32'b0;
        endcase
    end

endmodule
```

特别需要注意的是，自动生成的代码可能存在一定的 bug，需要大家仔细检查。但使用该插件仍然可以大大提高编写代码的效率。

4. 模块化设计与仿真调试（必读内容）

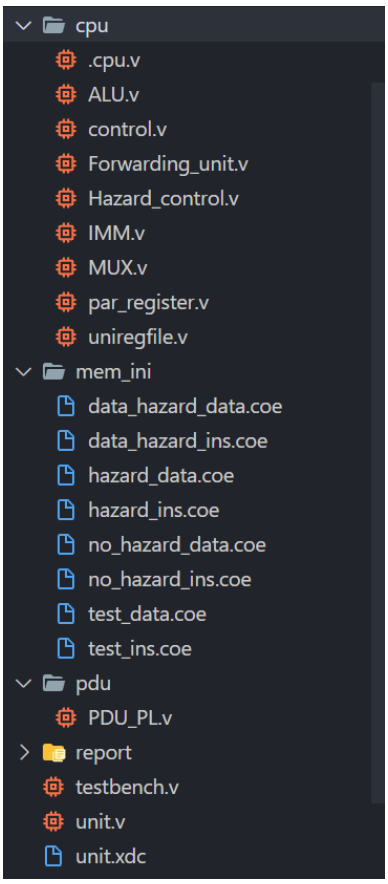
模块化，层次化设计是一种重要的硬件设计方法。它将一个复杂的系统分解为若干个子系统，每个子系统都是一个独立的模块，每个模块都有自己的输入输出。通过这种手段，我们可以将一个复杂的系统分解为若干个简单的模块，从而使得系统的设计更加简单，程序与代码也更加容易理解和维护，也就是降低了系统的 “Complexity”。（这一设计准则是硬软件设计领域中通用的）

“Complexity” 是大型项目开发中最大的障碍之一。它主要表现在系统的内部具有极高的耦合性（在软件上：比如全局变量的过量使用、面向对象中破坏类的封装从而暴露出不合适的接口；在硬件上：将时序与组合电路放入同一 always 块、各功能代码边界不明确，通俗的表现为大项目只有一个 module）。这种高度的耦合性使得系统内牵一发而动全身，一旦系统内部的某行代码甚至某个信号发生了变化，就会导致整个系统的行为发生变化。更糟糕的是，这种变化是难以预测的。你也不想你的代码依赖 bug 运行吧(笑。这就是过高 “Complexity” 的危害。

为了实现模块化设计，我们要将系统划分为合适的模块，每个模块承诺其自身对外提供的功能。当出现问题时，我们就只需要逐个检查各个模块是否正确地提供了承诺的功能，而不会漫无目的的修改一两句代码，祈求系统能够正常运行。

> 在自顶向下的模块化设计过程中，你甚至可以不具体实现各个模块，而是仅先给出各个模块输入输出的定义，先在顶层中将各个模块正确连接，最后再完成模块内部的具体实现。同时，模块内部也可以再细分为更多的子模块。在理想情况下，每一个最小模块提供一个最小的、不可再分的功能服务。

更为具体的设计方法会在后面的实验中加以详细介绍和反复运用。下面是助教在设计流水线时的功能分块（仅供参考）：

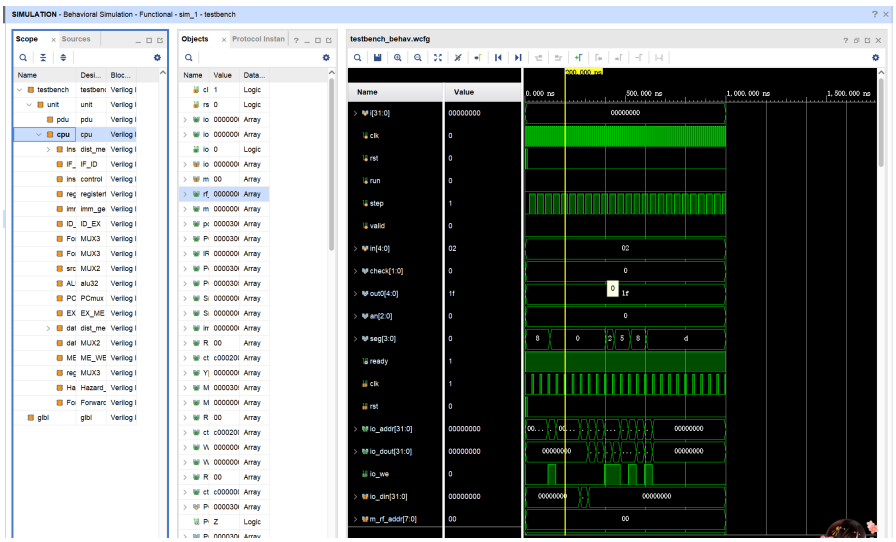


有如下两种仿真方式（任选一种即可）：

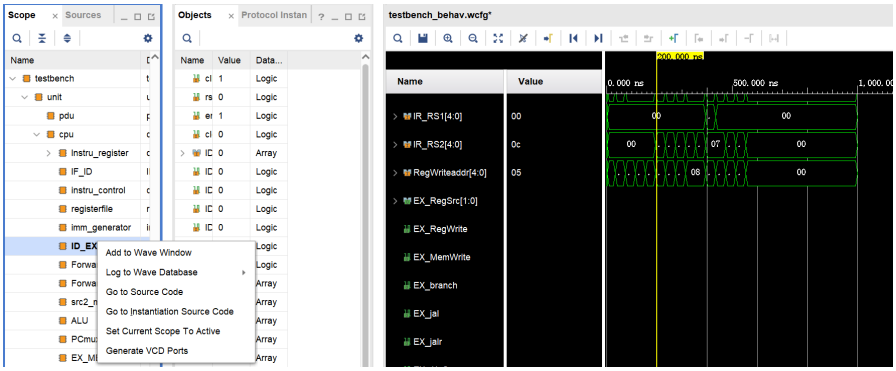
4.1 使用 Vivado 进行模块化仿真(必读内容)

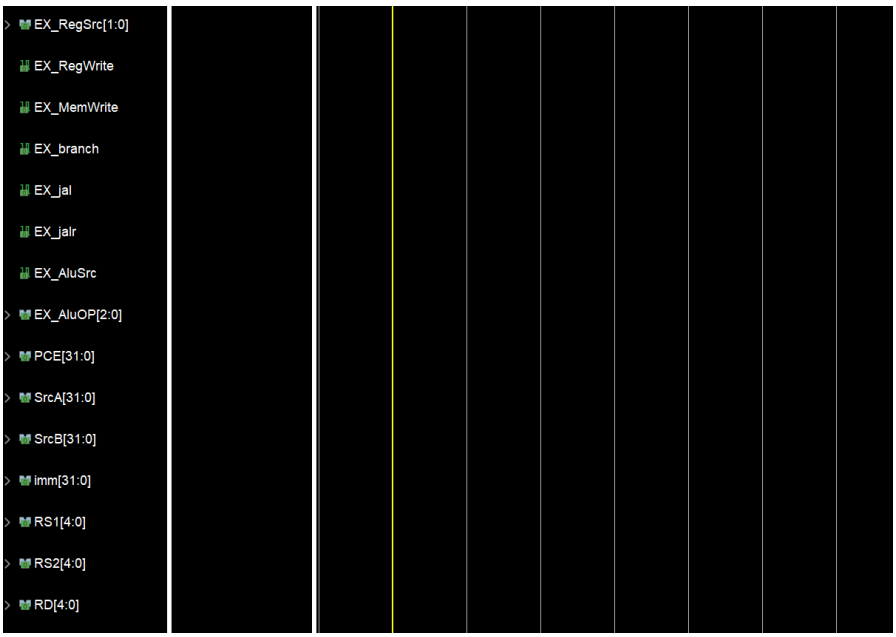
Vivado 为模块化的设计过程提供了强大的仿真工具，可以针对内部各个模块的输入输出绘制仿真波形。方式如下：

1. 添加好设计文件和仿真文件后，点击左侧的 “Simulate”，在弹出的界面中点击 “Run Behavioral Simulation”，即可开始仿真。此时仅能看到 topmudule 的输入输出波形，如下图所示：



2. 在最左侧的 scope 窗口中，右键你想要查看的模块。在弹出的界面中选择 “Add to Wave Window”，或者按住并将其拖动到右侧的 wave window 中，即可将其输入输出波形添加到当前 wave window 中，如下图所示：

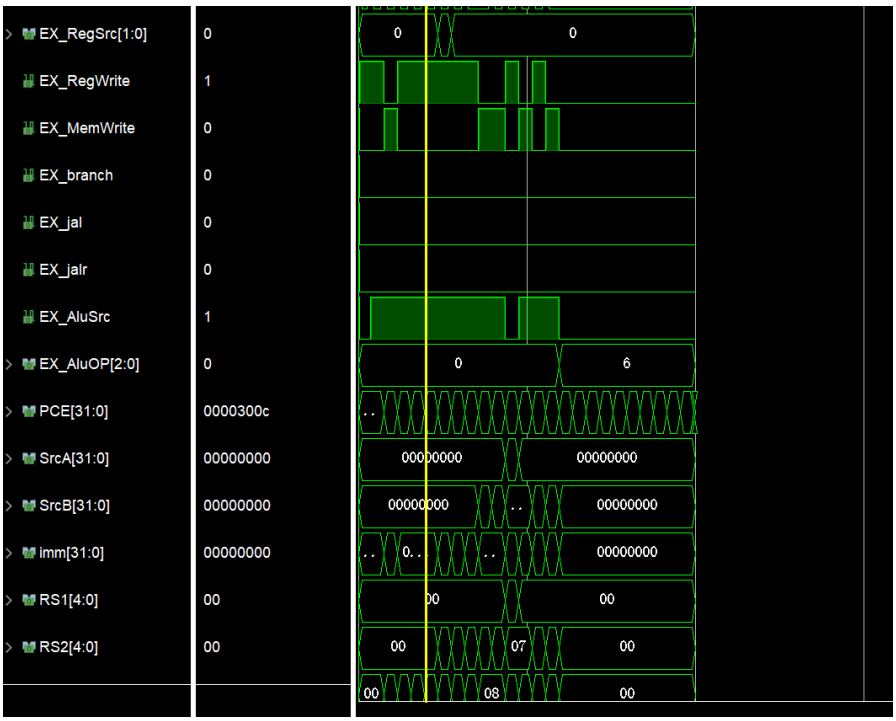




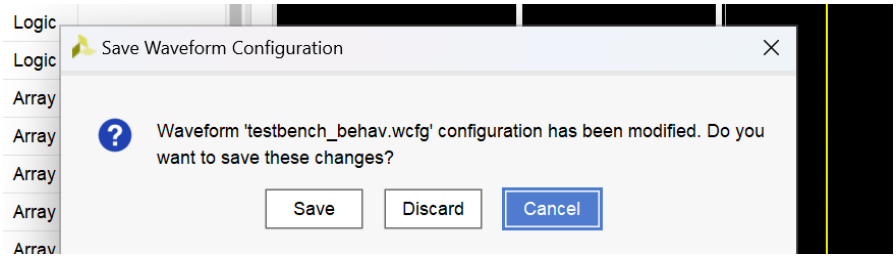
此时虽然信号添加到了 wave window 中，但是波形还是不可见的。

> 你也可以对中间一栏 “Objects” 中的信号进行同样的操作，这样就可以查看该模块中单独某些信号的波形，而不必将整个模块的所有输入输出加入波形图中。

3. 重新 “Run Behavioral Simulation”, 此时即可看到新添加的模块的输入输出波形，如下图所示：



请注意，在重新仿真时，在弹出的如下窗口中要选择保存配置，否则新添加的信号波形不会被仿真。



4.2 不依赖于Vivado的模块化仿真

> 提示：本方法不依赖 Vivado ，使用 MacOS 的同学可以参考。

1. 安装 iverilog ，安装方法可见上面的文档。

2. 安装 gtkwave

   在 MacOS 上，可以通过 brew install –cast gtkwave 来安装 gtkwave 。

3. 安装完上述软件后，就可以运行仿真了。以助教写的 testsim.v 为例：

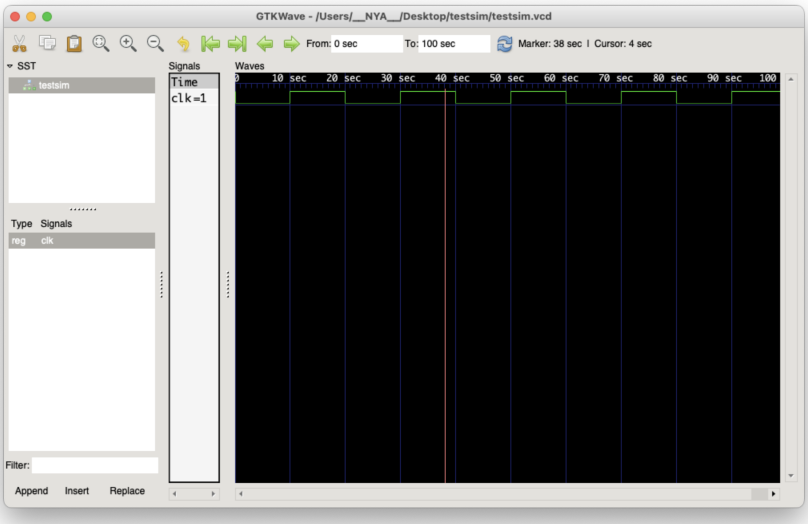
```
1module testsim ();
2    reg clk;
3    initial begin
4        $dumpfile("testsim.vcd");
5        $dumpvars;      // these two lines are necessary for generating .
     vcd file
6        clk = 1'b0;
7        forever #10 begin
8            clk = ~ clk;
9        end
10   end
11   initial begin
12       #100 $finish;   // one $finish flag is needed in this way
13   end
14endmodule
15
```

首先使用 iverilog -o testsim testsim.v 命令编译 testsim.v，得到的二进制文件可自定义命名；

然后使用 vvp testsim 命令，得到 testsim.vcd 文件（也就是 $dumpfile("testsim.vcd") 的作用）；

最后使用 open testsim.vcd，就可以使用 gtkwave 打开所得的 vcd 文件，查看波形。如图：

最后一步中， open 命令是 MacOS 特有的，别的系统可能需要用到类似“使用 gtkwave 打开”的方法来完成最后一步。



5. 实验题

光说不练假把式。请根据本文档列出的内容，结合自己所学知识完成下面的问题。

实验题 1. 在 Verilog oj47 中，我们实现了一个计数循环为 15 的计数器。事实上在此基础上，通过一些简单的修改，我们就可以实现一个任意位数的计数器。现在，我们希望完成一个时分秒时钟，该时钟在每次 clk 上升沿时秒位加 1（clk 信号不必以秒为周期），满 20 后清零，分位加 1；分位满 10 后清零，时位加 1；时位满 5 后三个位全部清零，如此循环。也就是说，我们设计的时钟一分钟只有 20 秒，一小时只有 10 分钟，一天只有五小时。

请尝试使用模块化的设计方法完成该时钟，并给出所有模块输入输出的仿真波形。

通常的设计方法可能是一个 always 里面使用 if 嵌套，分别对应秒位，分位，时位的计数，但是这样的设计一旦出现 bug，就很难定位到底是哪里出了问题（毕竟无法查看 always 块内信号的仿真波形），而且后续如果要添加新的功能如十分之一秒、天等单位也不利于扩展。

一个更好的设计方法是将时钟分为三个模块，分别为秒模块，分模块，时模块，每个模块其实就是一个独立的计数器。当出现 bug 或是需要增加新的功能时，我们就可以对症下药了。

Tips：你可能需要为单独的计数器添加额外的输入输出端口以处理进位问题。

下面是我们提供的示例框架代码：

```
1       module Clock (
2           input clk,
3           input rst, // asynchronous reset, active high
4           output [2:0] hour,
5           output [3:0] min,
6           output [4:0] sec
7       );  // you should not change code upon this line
8           // your code here
9           // you may need to add some extra signals
10
11          // think the ports needed and how to connect them
12          Sec sec1();
13          Min min1();
14          Hour hour1();
15      endmodule
16
17      // implement the three modules here
18
```

注：你不必严格按照上面的代码来实现，但需要确保 module Clock 的输入输出端口与上面的代码一致。

助教为你提供了 testbench 文件（在bb平台），请使用该 testbench 进行仿真（你需要将Vivado仿真的时间上限调整到足够长，以便能至少看到一次时钟的完整运行）。

你需要提交的内容应当包括：

1. 你的代码； （1 point）
2. **所有模块**（Clock，Sec，Min，Hour）的输入输出端口仿真波形； （1 point）
3. 运行 RTL Analysis 任务下的 Open Elaborated Design。请提交使用 Vivado 生成的电路图； （1 point）

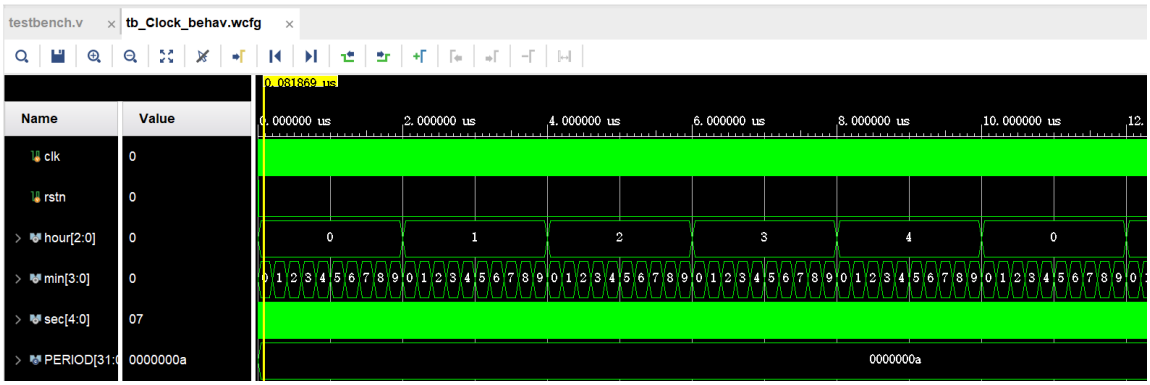
4. 实验反馈。欢迎大家根据对本文档给出自己的感受和建议。

> 本学期实验文档将由助教们精心设计和重写，大家的反馈对于我们来说非常重要. 我们会根据大家的反馈来改进后续实验设计和文档。

> 请务必不要在always时序电路中使用阻塞赋值，这样的提交会被扣除额外分数。它的危害可见FAQ部分。

> 本次实验并不是我们的正式实验内容，因此不会影响实验总分。最后的实验题会作为第三次作业的内容，占三分。本实验题提交的截止日期为 3 月 26 日晚 23:00:00，也就是下个星期天（大家不要熬夜赶实验哦）。建议将所有内容附在一份 pdf 文档中，命名为“学号_姓名_lab0.pdf”，并将该 pdf 文档提交到 bb系统。截止时间之后的提交最高得分为 1 分。
> 第三次作业提交时间和方式不变，我们将在下周三课后布置。

附：CLock 的仿真波形参考：



6. FAQ, 配置有问题先看这里

实验中可能会遇到一些问题，这里列出了一些常见的问题和解决方法(会经常更新)。

https://cscourse.ustc.edu.cn/vdir/Gitlab/PB20020586/lab-of-cod-faq/-/blob/master/lab0FAQ/lab0FAQ.md