

Episode 1 Nice to meet you, Vivado!

——《你可能不知道的verilog提高班指南》By TA-马子睿

欢迎各位新同学来到数字电路实验提高班！即日起，我将不定期发送一些关于本实验课程的提示与指南，希望能够帮助大家实验中轻松破局、无往不利！

在本集中你可以看到：

- Vivado的使用方法
- Vivado的各类功能

目录

- Vivado的使用方法
 - [Vivado的下载](#)
 - [创建项目中的硬件配置](#)
 - [如何完成一个完整的工程](#)
 - [创建设计资源](#)
 - [RTL分析](#)
 - [Simulation](#)
 - [综合和实现电路](#)
 - [生成比特流文件并烧板](#)
- Vivado的使用技巧
 - [更好的编辑器](#)
 - [更快的编辑速度](#)
 - [更便捷的仿真信号](#)
- [结语](#)

Vivado的使用方法

Vivado的下载

先为大家推荐仍然可以使用的Vivado2021百度网盘下载源：

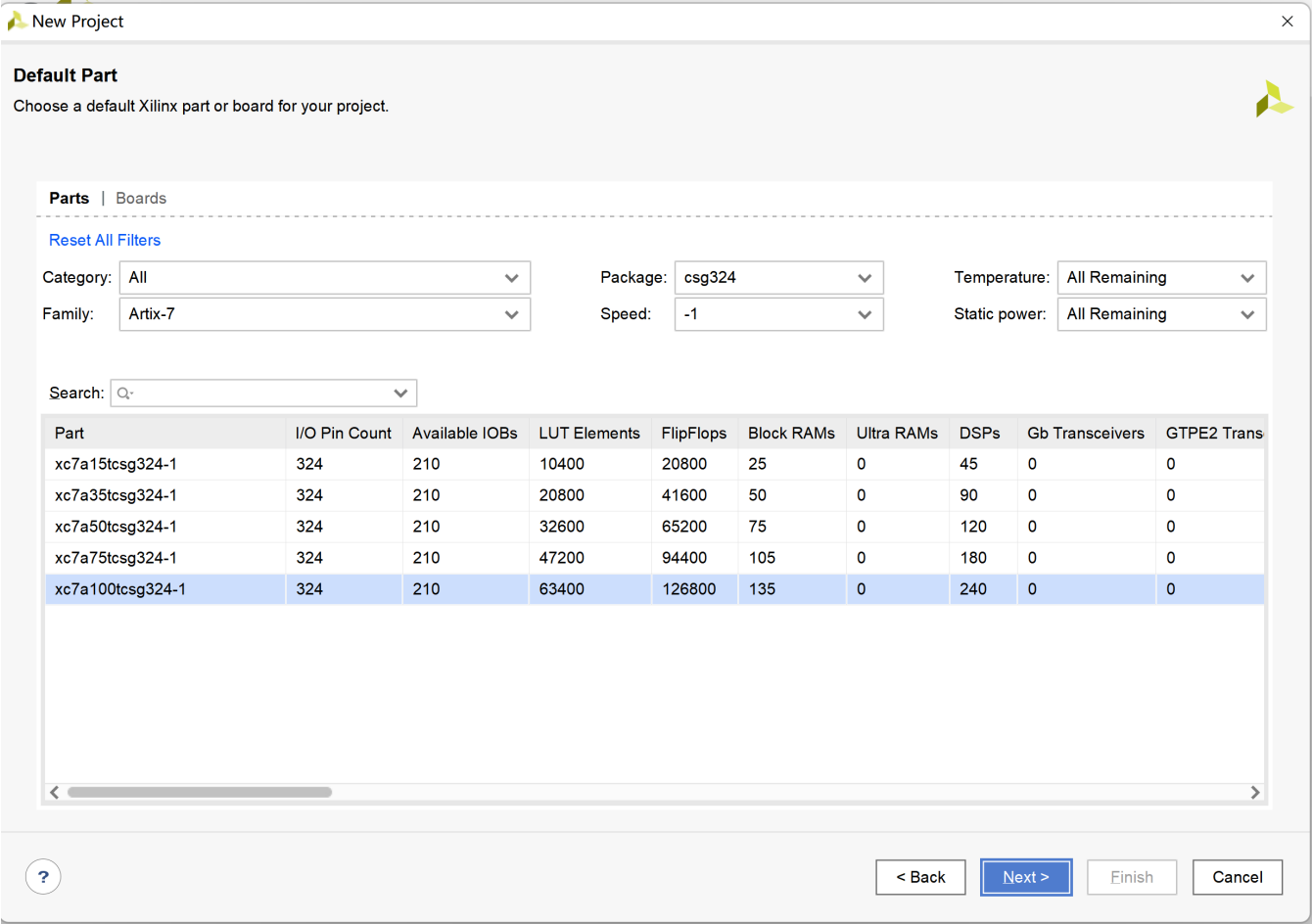
链接：<https://pan.baidu.com/s/19beEGEPSzOAggGFtPjw6Mw>

提取码：DZKR

推荐使用vivado2021的原因：

verilog本身是一个很容易出现在书写过程中出现warnings的语言，这些warnings只能由vivado报出（编辑器无能为力），但2019及之前的版本动辄几千个warnings，大部分还是完全没有用的，这很容易让真正的warnings藏于其中难以发现。但vivado2021报出的warnings一般是真正的警告，95%的警告确实是需要改动代码的。

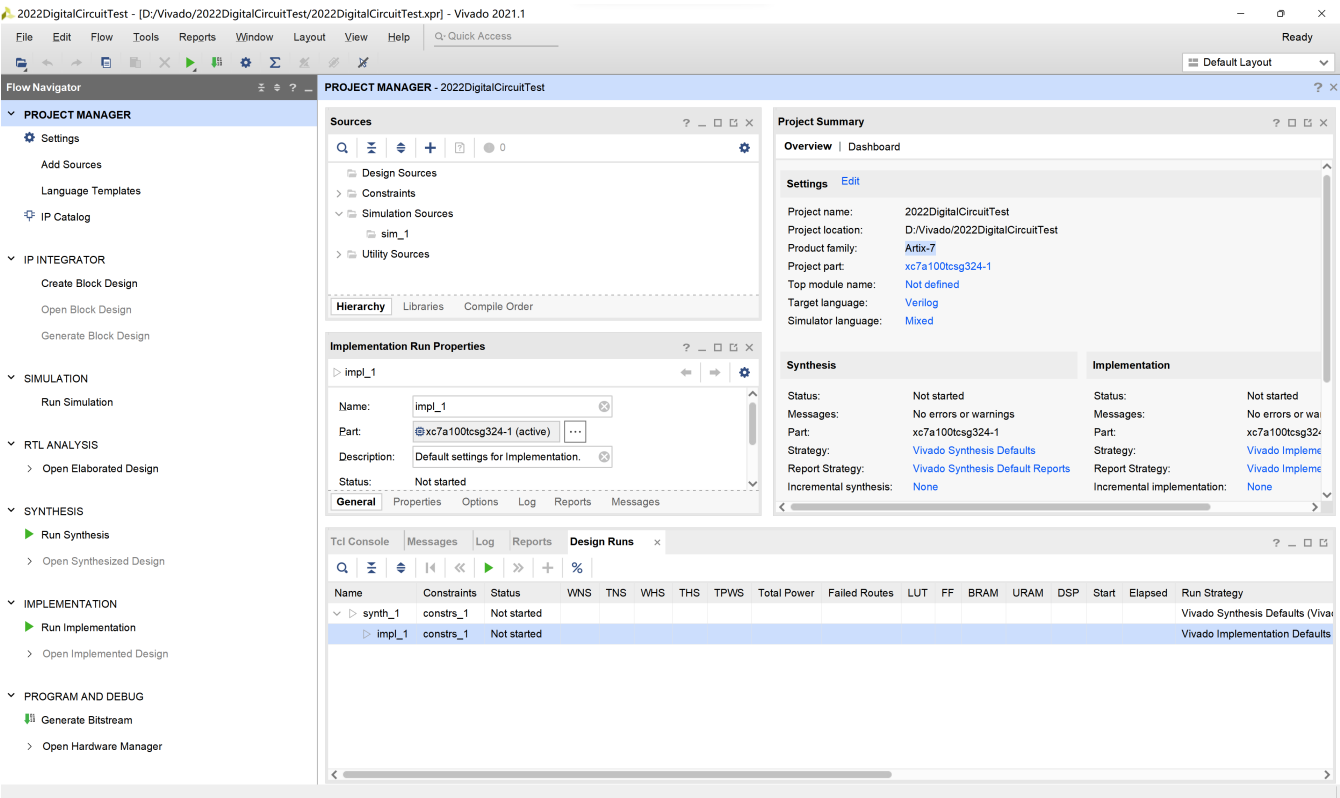
创建项目中的硬件配置



上方的搜索栏可以在其中输入如图的配置，就可以找到这一款开发板啦！

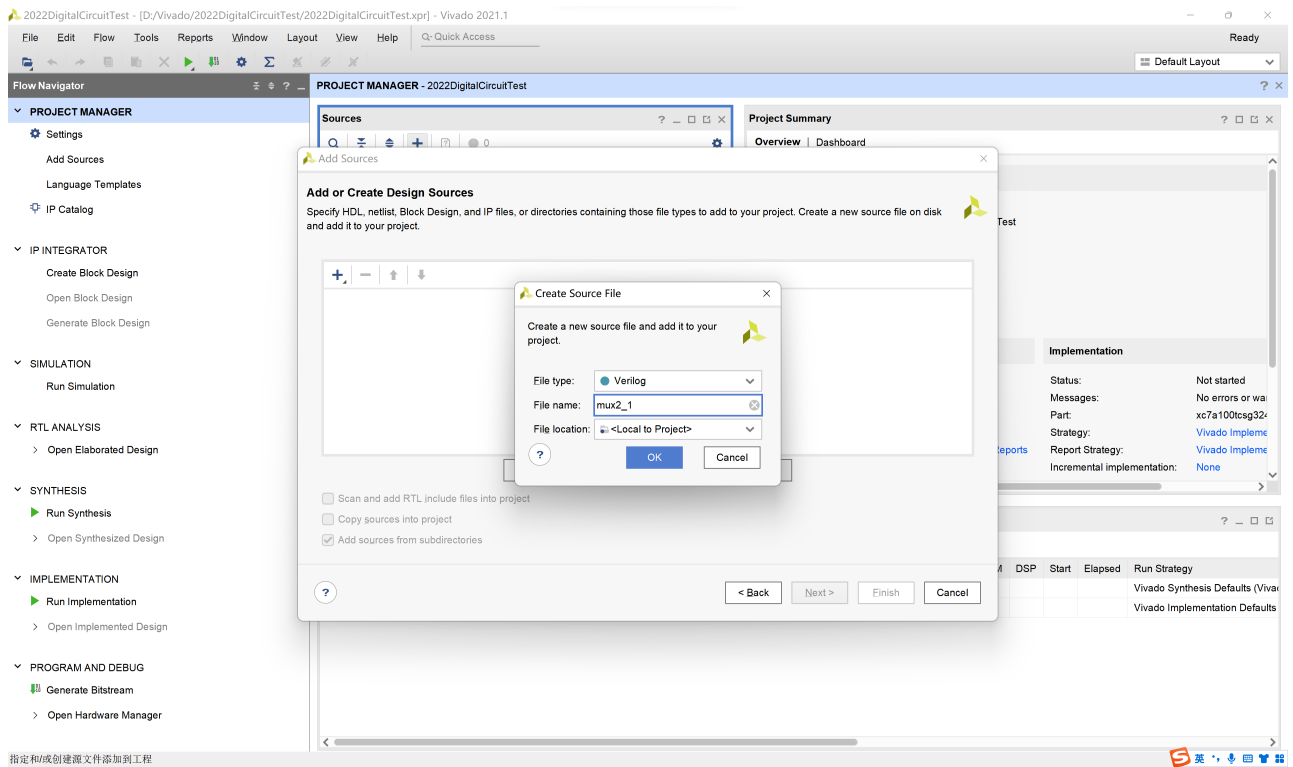
如何完成一个完整的工程

进入到Vivado中，是这样的界面

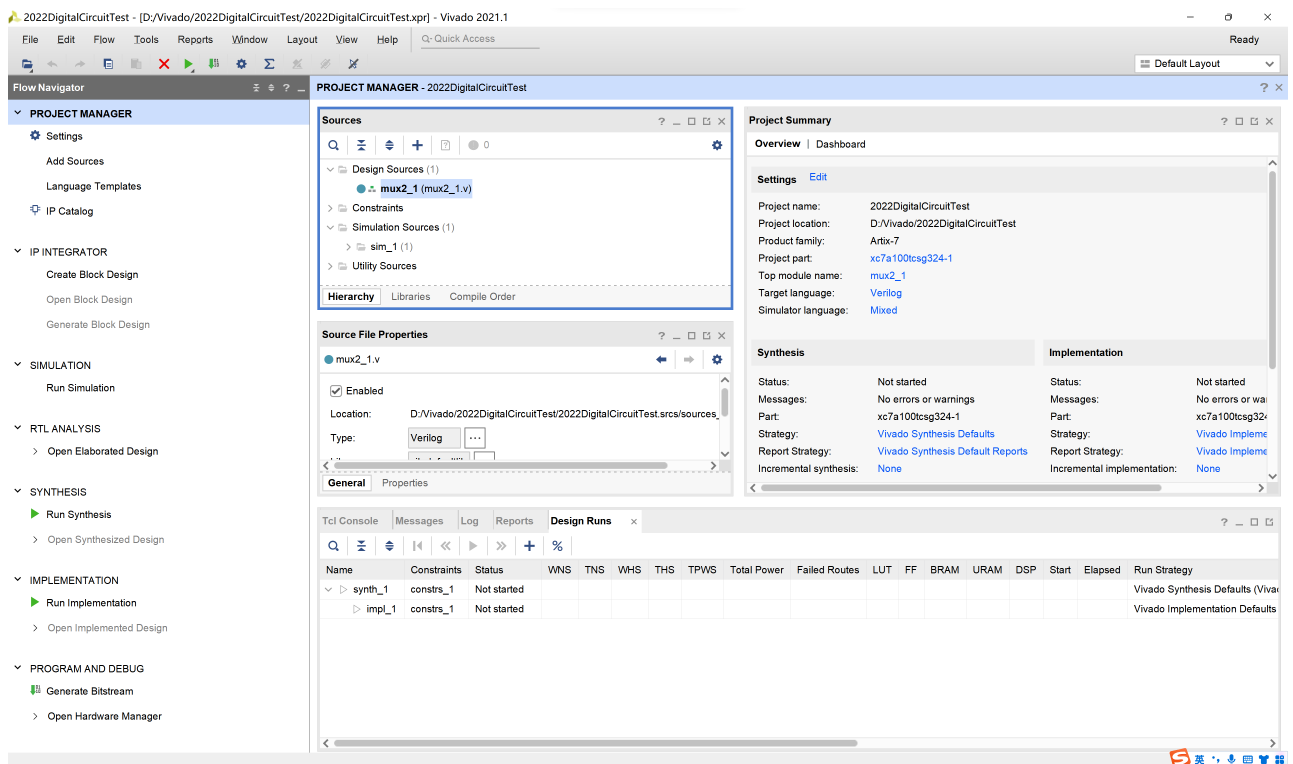


创建设计资源 (design sources)

- 点击Add Sources，即可创建一个全新的设计文件



- 创建后，我们就可以双击这个文件，在编辑器中编辑它了！



一个32位宽2选1选择器的参考代码如下：

```
module mux2_1(  
    input    [31:0] din1,  
    input    [31:0] din2,  
    input    sel,  
    output reg [31:0] dout  
);  
  
always @(*) begin
```

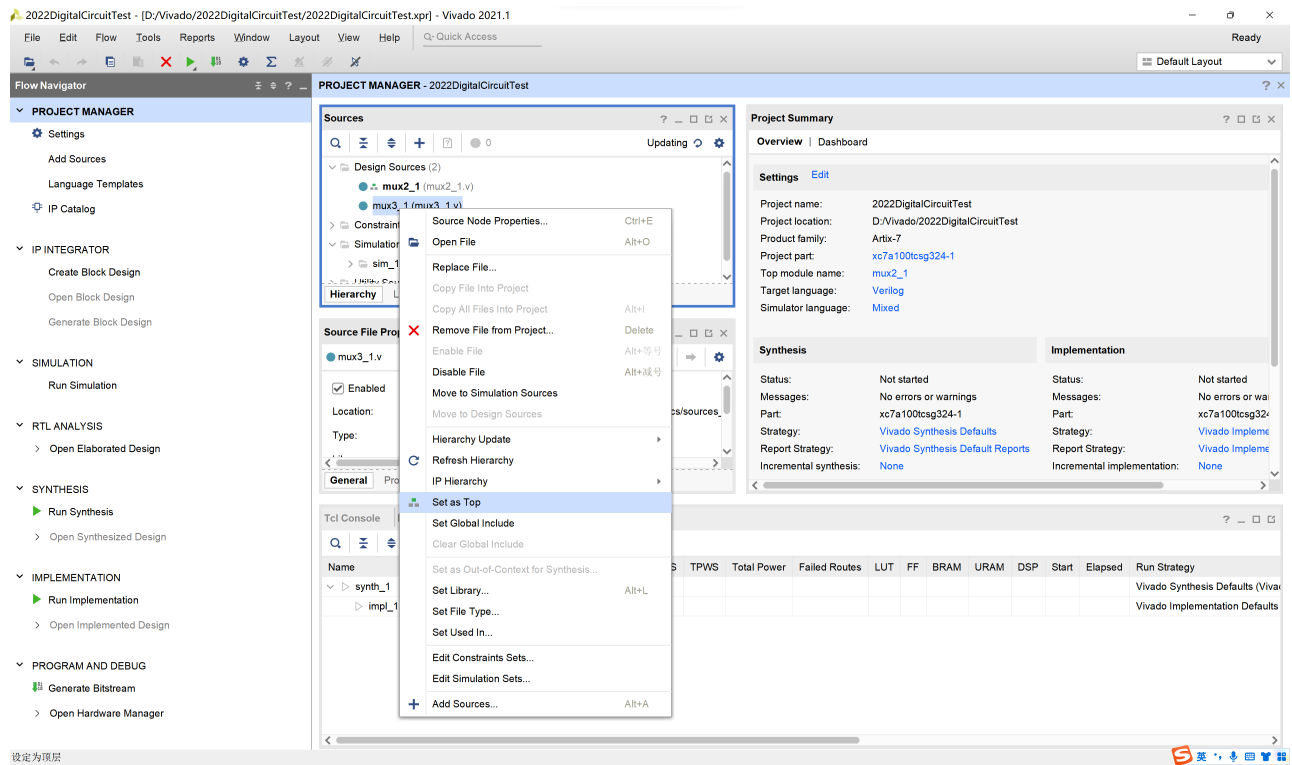
```

        case(sel)
        1'b0: dout = din1;
        1'b1: dout = din2;
        endcase
    end
endmodule

```

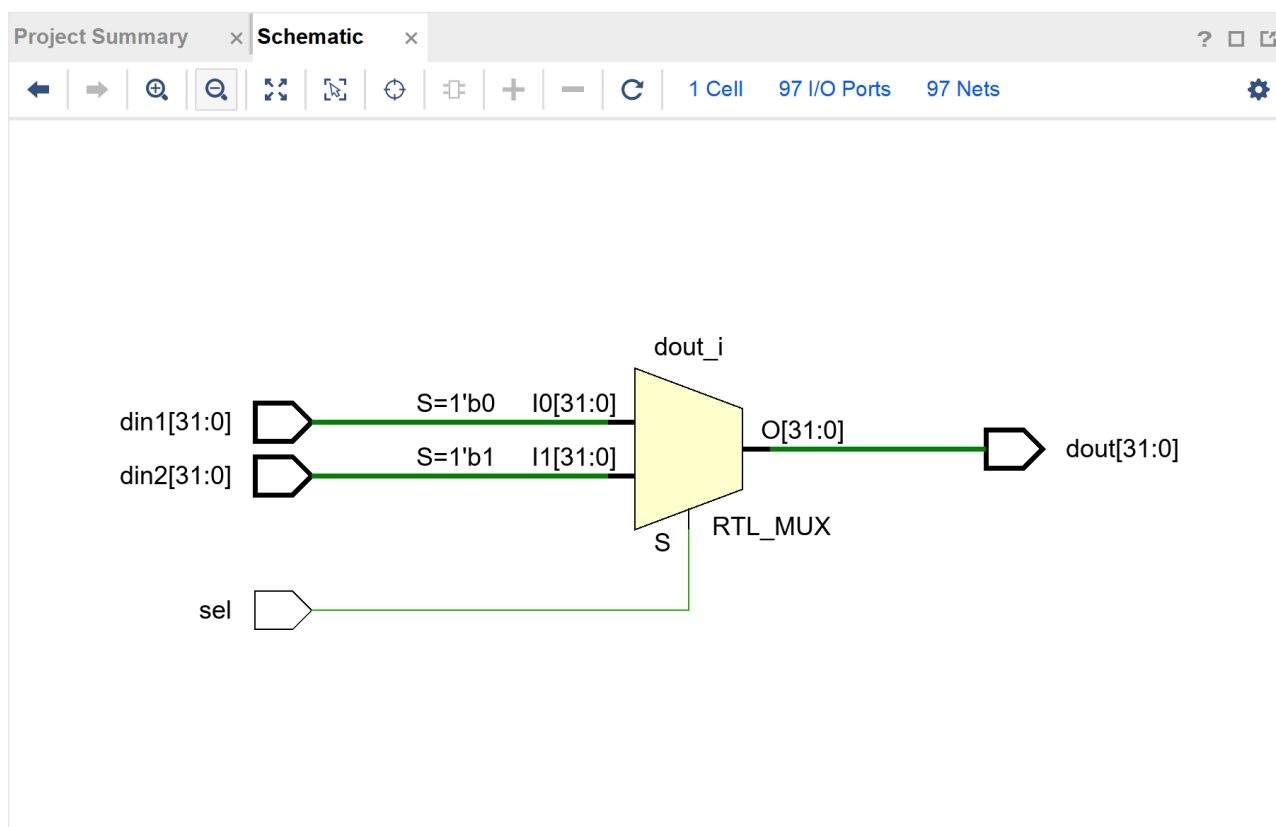
- 完成这个代码后，我们肯定希望看看它画出的电路图是怎么样的，这里我们需要使用**RTL分析**。

这里需要注意，之后所有的操作**仅仅对Top文件有效**，如果你希望画出另一个工程文件的电路图，**需要将它设置成Top文件哦！**



RTL 分析

- RTL分析可以将verilog代码转换为基本的电路图：



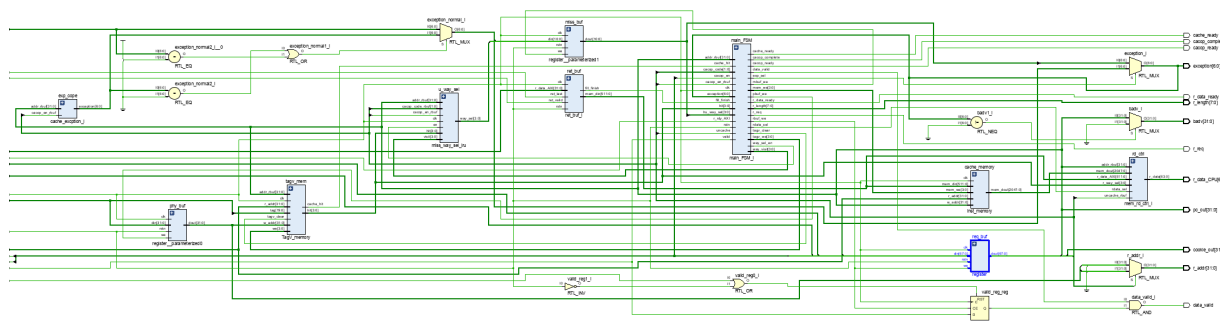
可以看到，一个二选一选择器已经被描述出来了！

- RTL分析的作用有哪些呢？

1. **可以大致看出所描述的电路是不是按照自己的想法描述的**。这里需要指出的是，**并不是所有的设计都是按照想法实现的**，特别是对于编码器、译码器，可能在RTL电路中会使用一个ROM（只读存储器）来查找实现。
2. **可以看出是否有的线路被空接或位宽不匹配**。这样的问题也可能在warnings中被报出，但RTL电路无疑是一个更加直观的检索方法。
3. RTL分析如果失败，**可以帮助我们快速定位代码中是否有严重错误**，比如把**always**块中被赋值变量不是**reg**、**reg**类型被用于例化输出接口，等等。

- RTL分析自然也有它的局限性：

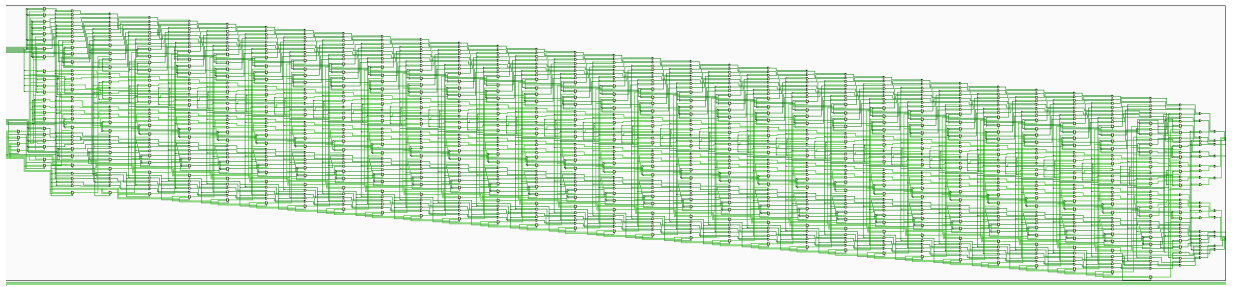
1. 分析出的电路并不是最后真正的电路，比如RTL可能会给出一个八选一选择器，这显然是不能直接实现的（可能会用三级2选1实现）。
2. RTL电路图不会按人类的想法画，换句话说，它会把元件的位置乱摆，让大家难以直观理解：



（图源：2022第六届“龙芯杯”大赛——中国科大“小步快跑”队LoongarchCPU设计之指令高速缓存模块）

图中一个按照人类理解应该画在最左边的缓冲被画在了最右边（图中被蓝框选中的部分）

3. 如果使用了一些高级语法，比如for循环语句，那么即便只有几行代码，RTL电路的长度也可能超乎你的想象：

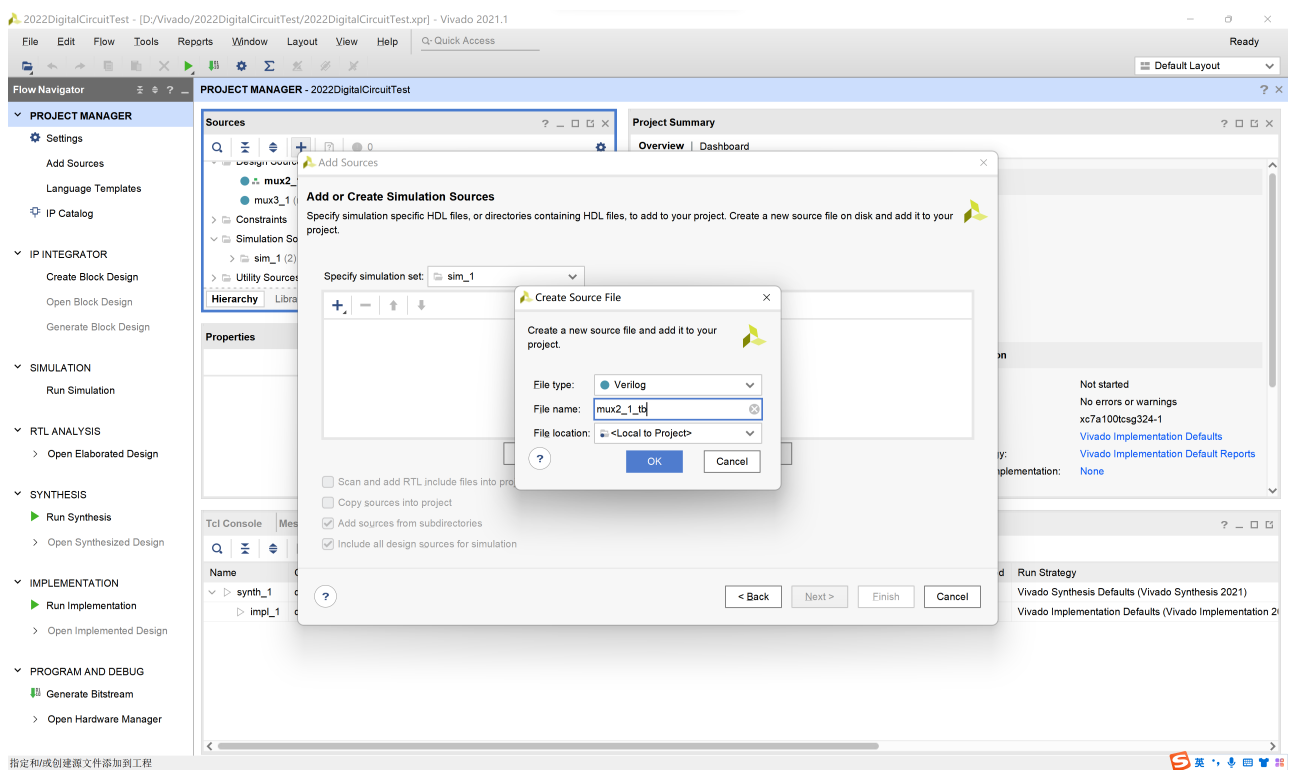


(图源：2022第六届“龙芯杯”大赛——中国科大“小步快跑”队LoongarchCPU设计之TLB信号生成模块)

- 需要注意的是，如果你通过RTL看出了问题并修改，之后必须要关闭RTL再打开，方可看到新的警告或错误（若直接点击重新生成，则不会报错）

Simulation 仿真

- 在工程正式上板之前，应当正确使用仿真来避免烧板的大错误，防止损坏开发板。
- 首先，我们应当创建仿真资源（也就是仿真文件）：一般习惯上，我们在需要测试的模块名后加_tb来表示testbench仿真文件



这里给出测试的一个样例：

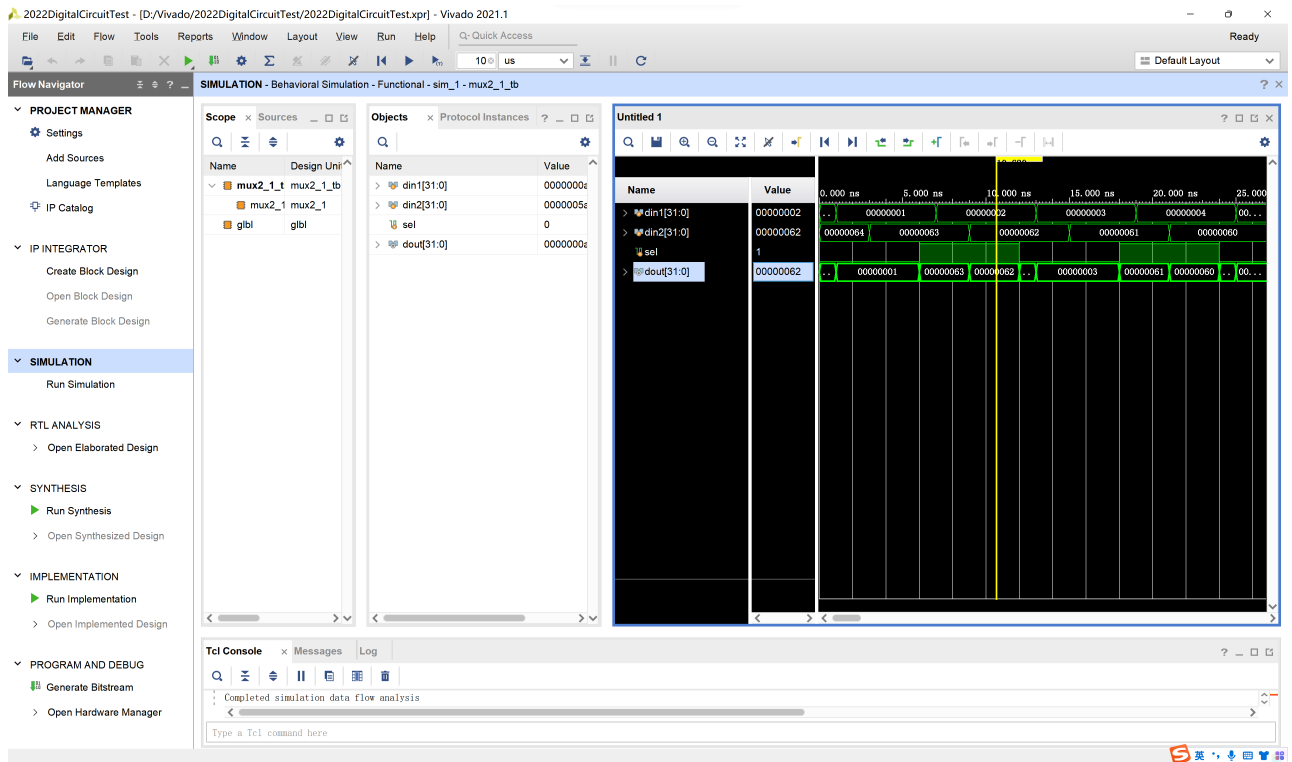
```
module mux2_1_tb(  
  
);  
    reg [31:0] din1, din2;  
    reg sel;  
    wire [31:0] dout;  
    mux2_1 mux2_1_sim(  
        .din1      (din1),  
        .din2      (din2),  
        .sel        (sel),  
        .dout       (dout)  
    );  
    initial begin  
        din1 = 0;  
        din2 = 32'd100;  
    end  
endmodule
```

```

sel = 0; // 对三个信号进行初始化
repeat(10) begin // 重复十次
    #1 din1 = din1 + 1; // 每隔一个时间单位，令din1自增1
    #2 din2 = din2 - 1; // 每隔两个时间单位，令din2自减1
    #3 sel = ~sel; // 每隔三个时间单位，令sel取反
end
end
endmodule

```

在仿真文件中，我们应当例化我们所需要测试的功能单元（这里是mux2_1，有关例化的更多技巧详见后续文章），之后使用repeat、forever等语句实现输入信号按照时间变化的仿真过程。仿真结果如下：



以上是行为仿真模式，除此之外，还有时序仿真，可以用来对逻辑延迟进行仿真。不过，对于当前较为简单的设计，时序仿真的意义暂时体现不出。

综合和实现电路

• 综合电路 (Synthesis)

1. 在本步骤中，Vivado将会把描述的电路正式编译出来。在这里，我们可以找到更多warnings、critical warnings和errors，例如**逻辑环路**（自己的输出直接接到自己的输入）、**多驱动**（一个变量由多个always块或多个输入修改）等。
2. 综合电路也可以查看电路图，**但这时的电路图大部分以查找表出现，很难找出问题。同时，综合出的电路也有可能把某一模块的电气元件安置到另一个模块里**（为了更优的电路性能），所以大家在使用综合出的电路图查找问题时一定要关注到这些问题

• 实现电路 (Implementation)

1. 本步骤需要一个全新的文件：限制文件。不同的开发板所需的限制文件不同（**本课程实验所需的限制文件请见群中文件**）。在使用文件时，请将需要连接的接口的注释符号去掉，改好对应接口的名字即可。**这里由于板上外设不够，我们只能改为实现7位数的多选，如下：**

```
##Switches

set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { din1[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { din1[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { din1[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { din1[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { din1[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { din1[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { din1[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { din2[0] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { din2[1] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { din2[2] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { din2[3] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { din2[4] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { din2[5] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { din2[6] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { sel }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
# set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { din2[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs

set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { dout[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { dout[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { dout[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { dout[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { dout[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { dout[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { dout[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
```

2. 在实现电路时，更多更难以解决的问题将浮现，比如时序不满足（逻辑电路过长）等。但同时我们也可以看到更多有关资源使用（utilization）、功耗（power）、时序（timing）的信息：

Tcl ConsoleMessagesLogReportsDesign RunsPowerDRCTimingUtilization x

QZ%

Hierarchy

Hierarchy

Summary

Slice Logic

Slice LUTs (<1%)

LUT as Logic (<1%)

Slice Logic Distribution

Slice (<1%)

SLICEL

Name	Slice LUTs (63400)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)
N mux2_1	4	3	4	22

utilization_1

Tcl ConsoleMessagesLogReportsDesign RunsPower x DRCTimingUtilization

QZC

Summary

Settings

Summary (7.365 W, Margin: N/A)

Power Supply

Utilization Details

Hierarchical (7.217 W)

Signals (0.125 W)

Data (0.125 W)

Logic (0.018 W)

I/O (7.074 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 7.365 W

Design Power Budget: Not Specified

Power Budget Margin: N/A

Junction Temperature: 58.6°C

Thermal Margin: 26.4°C (5.7 W)

Effective θ_{JA} : 4.6°C/W

Power supplied to off-chip devices: 0 W

Confidence level: [Low](#)

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

On-Chip Power

Dynamic: 7.217 W (98%)

98%

97%

Signals: 0.125 W (2%)
Logic: 0.018 W (<1%)
I/O: 7.074 W (97%)

Device Static: 0.147 W (2%)

impl_1 (saved)

生成比特流文件（.bit）并烧板

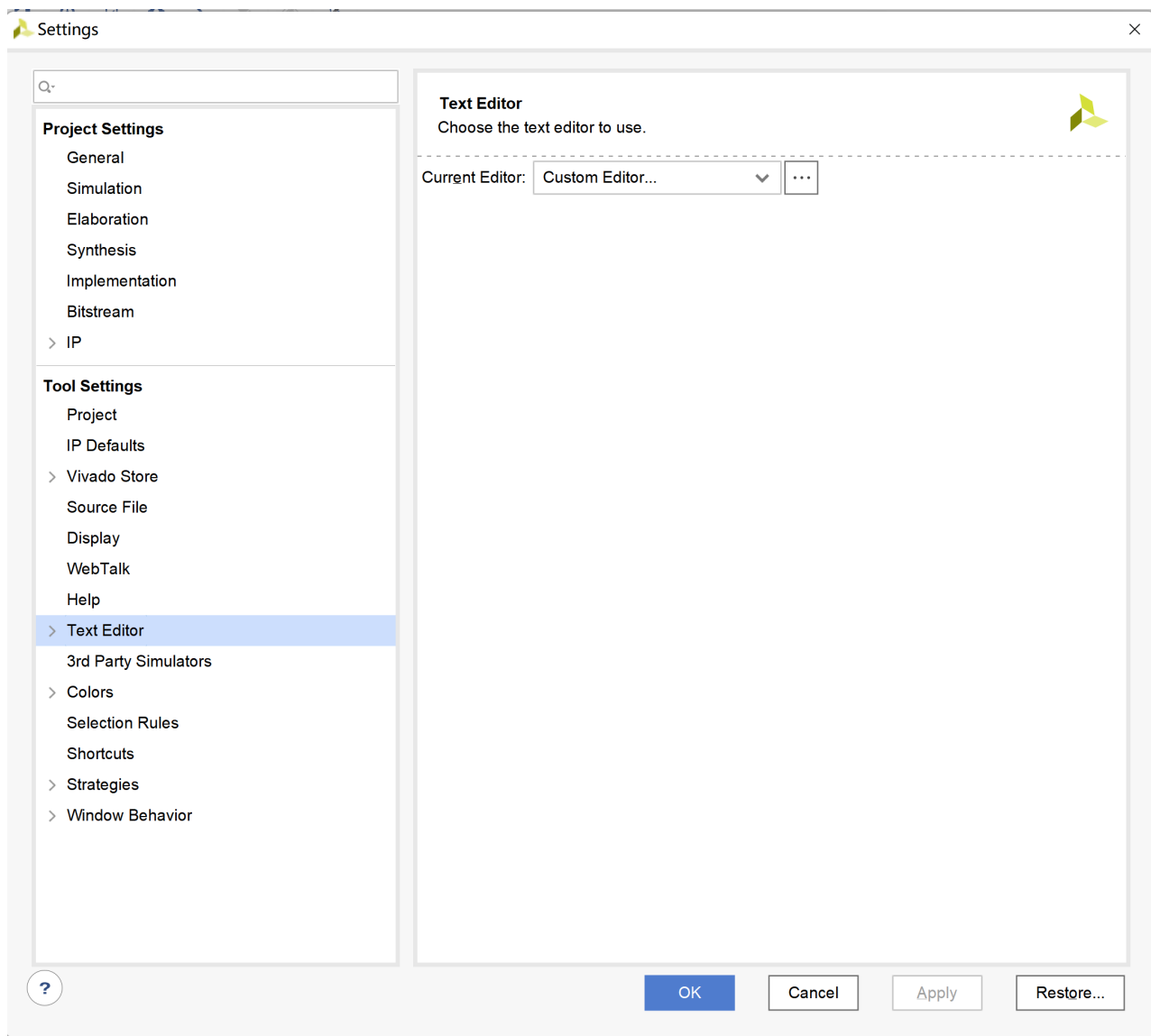
- 点击Generate Bitstream即可生成比特流文件，**建议每次生成后将其拷贝出来，防止下次综合时清除掉原先的比特流文件**
- 打开 HardWare Manager 对刚刚生成的比特流文件进行烧板，之后就可以通过开发板来进行操作啦！

Vivado的使用技巧

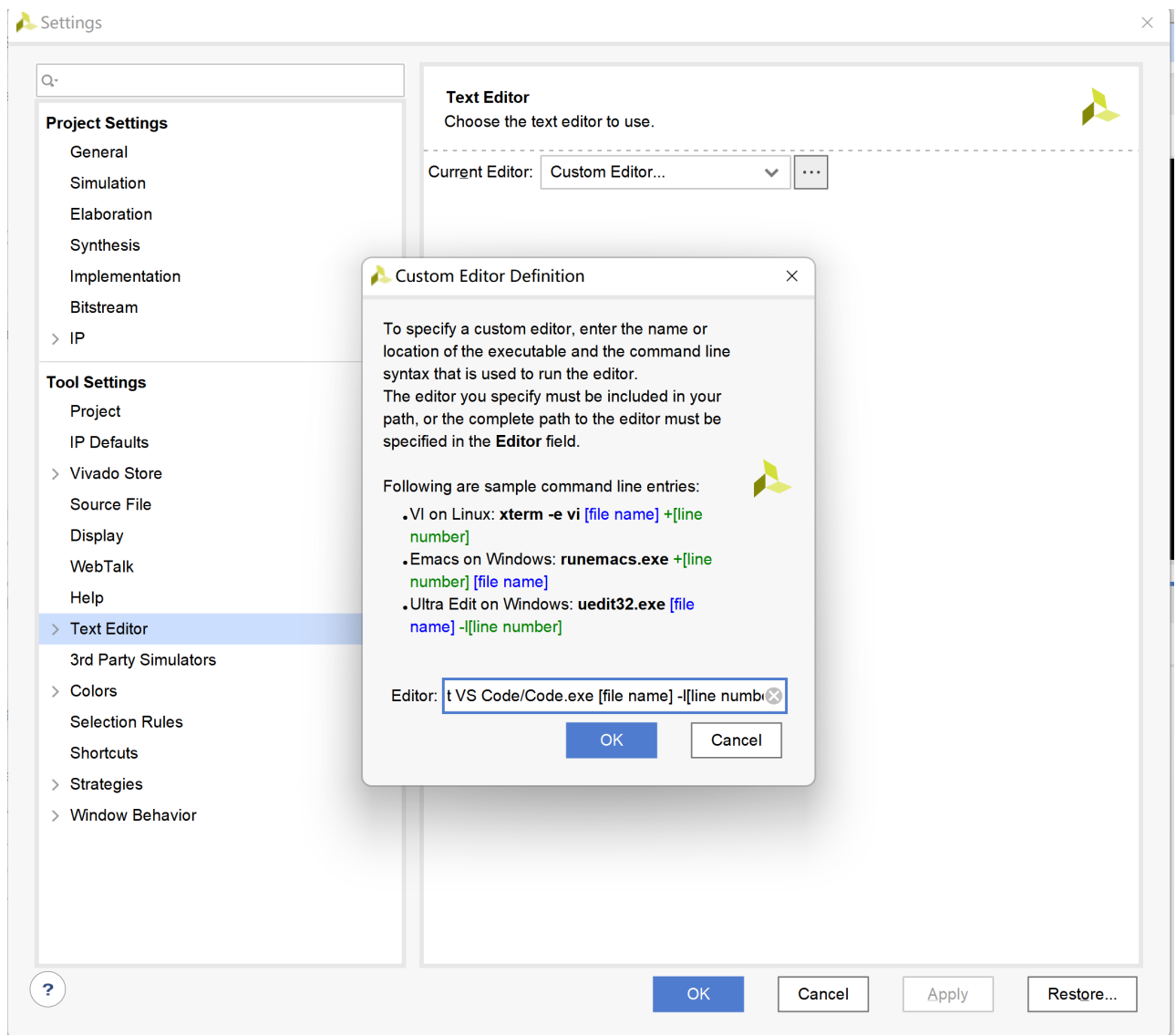
更好的编辑器

Vivado自带的编辑器虽然可用，但依然不及强大的vscode。如何使用vscode在Vivado内编辑代码呢？请跟着我做如下步骤：

- 点击Vivado菜单栏的Tool，选择Settings打开设置界面，选择Text Editor：

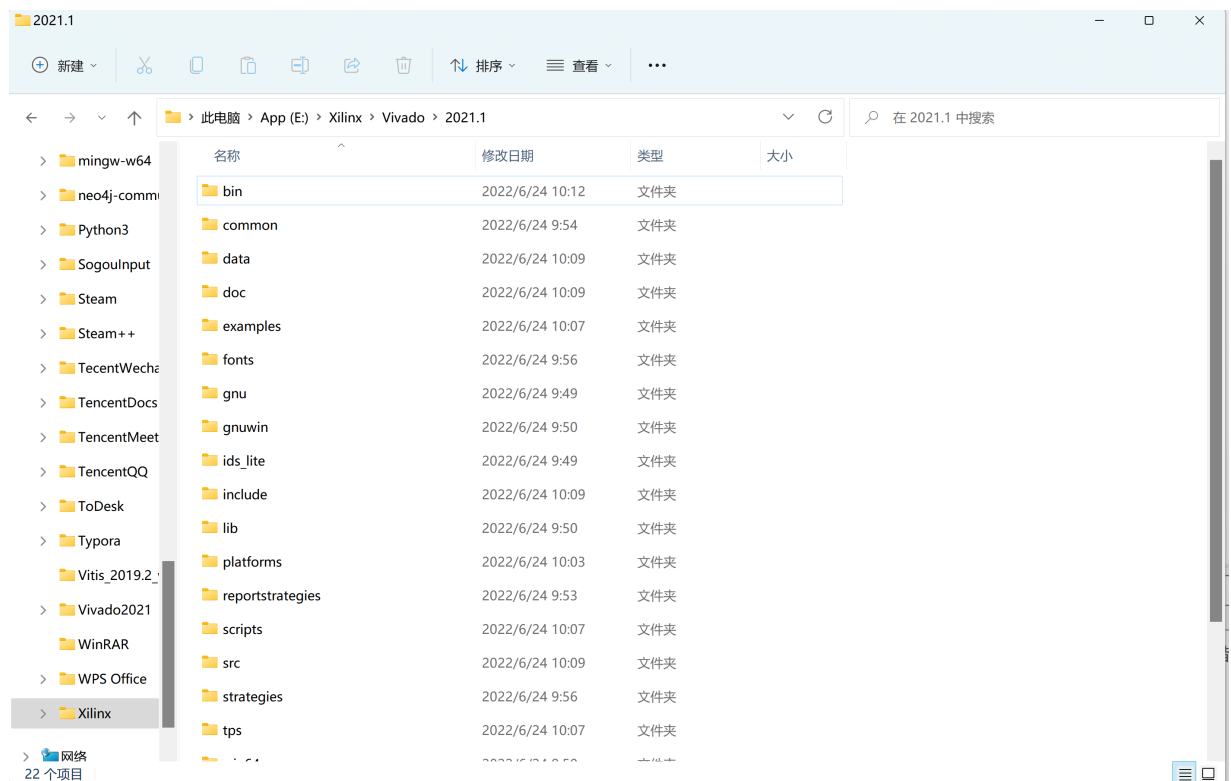


- 点击右侧省略号，按照要求填写编辑器路径和必要的补充：

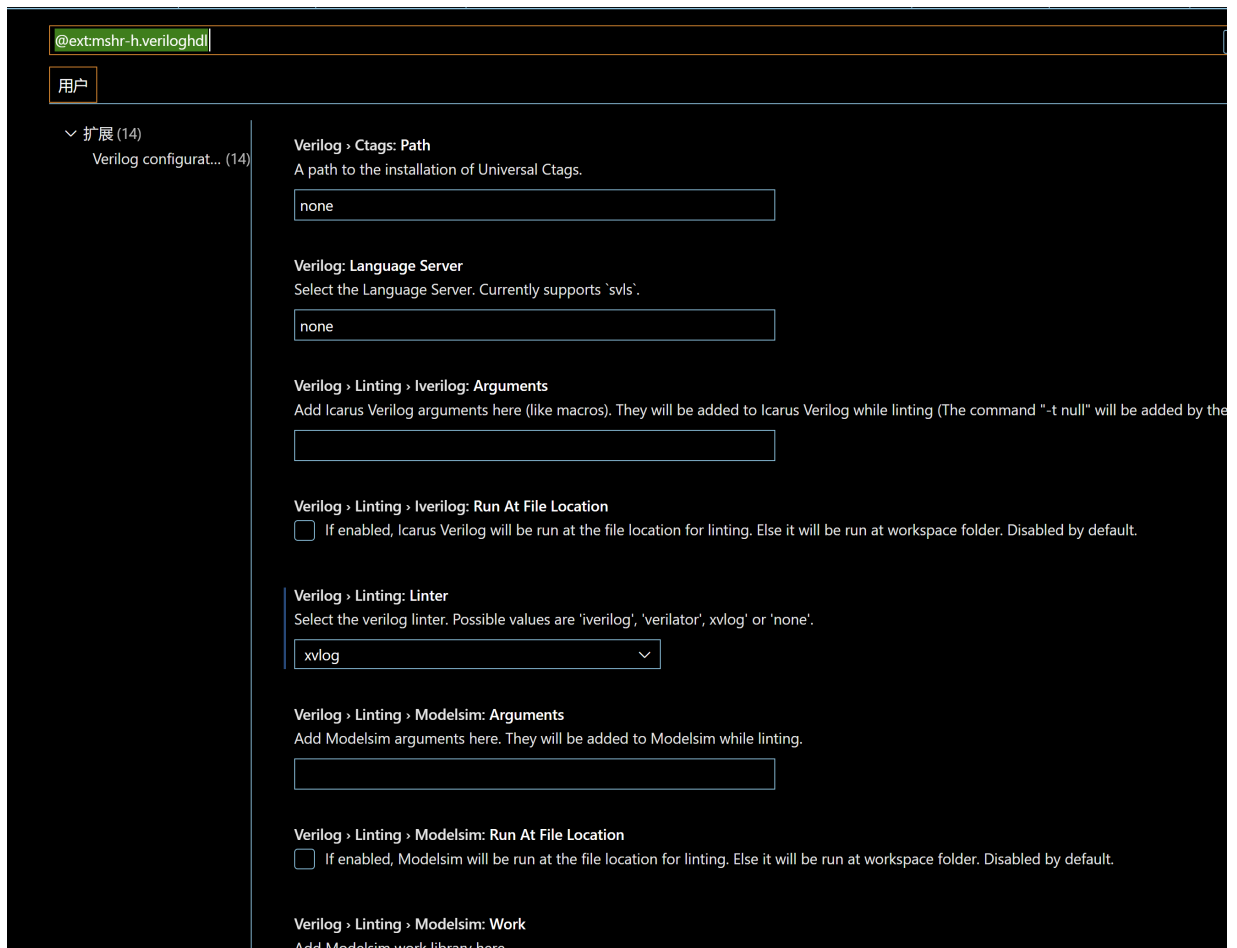


之后就可以使用vscode进行编辑啦！可以在vscode中安装必要的verilog拓展。如果想要让vscode可以在编辑时自动报出语法错误，你需要：

- 打开“自动保存”功能
- 将Vivado安装目录下的bin文件夹放入环境变量中。大致路径为：



- 在vscode中安装Verilog-HDL/SystemVerilog/Bluespec SystemVerilog拓展包，之后点击这个拓展包的扩展设置，找到Linter，选择为xvlog，即可激活自动报错功能：



更快的编辑速度

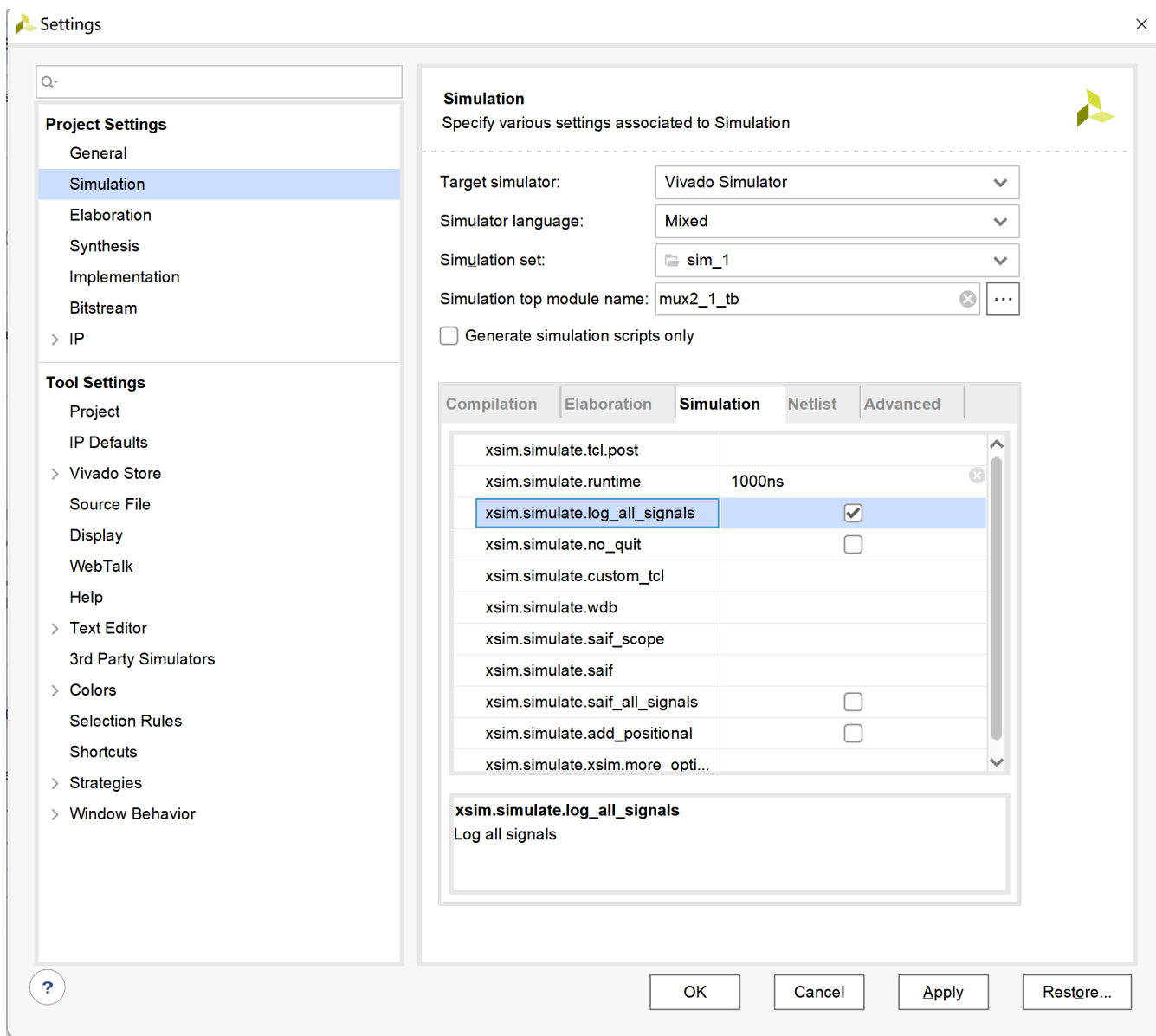
在verilog中，经常会遇到对同一列的一系列编码或变量名进行改动的情况（这里不点名限制文件了）。使用vscode，可以让多点编辑变得简单。

- 使用Alt，之后使用鼠标多点几处，就可以同时编辑这些内容啦！
- 使用Alt+Shift，用鼠标点击同一列的不同行，就可以选中这些行的同一列位置进行编辑。

请善于使用这两个快捷键，可以助你事半功倍哦！

更便捷的仿真信号

在设置中，可以设置“记录全部仿真信号”。这是因为默认情况下，仿真器只会记录输入、输出信号，对于中间量的信号则不会记录。只需要在设置中这样设置就好：



这样一来，我们需要查看某个内部信号时，就无需拖入仿真信号图后重新仿真了，只需要拖入仿真信号图即可看到全部信号！

结语

在本辑中，我们介绍了Vivado的一些基本应用，以及如何完成一次工程。除此之外，大家也可以对settings里面的各种设置进行探索，如果有更多更好的发现欢迎告诉助教哦！

另外，若本辑中的说明有误，也欢迎各位同学不吝赐教！