北京工业大学信息学部-软件学院2022-2023学年第一学期

**《嵌入式微处理器结构与应用》**

**CPU设计实验报告**

**班级 200803**

**学号 20080309**

**姓名 常兴阳**

# CPU设计规格

本次实验中所设计的CPU具有以下规格：

1. 一个全功能的CPU，可以在FPGA上综合和仿真；
2. 保证操作正确性的前提下拥有尽量简单的结构；
3. 实现此设计所需的Verilog源代码数量较小
4. 本CPU可以用汇编语言进行编程，由于没时间编写针对于C语言转汇编的编译器，故现不支持C语言编程
5. 使用精简完整的指令集，包括中止(HALT)，压栈立即数(PUSHI)，按址压栈(PUSH)，弹出栈(POP)，立即跳转(JMP)，栈顶值为0跳转(JZ)，栈顶值为1跳转(JNZ)，外部输入(IN)，输出外部(OUT)，操作(OP) 一共10条指令;
6. 有一个至少16位的体系结构；
7. 支持通过指定方式输入/输出；
8. 支持一般的条件语句操作，通过JMP、JZ、JNZ实现；
9. 采用简洁的堆栈体系结构，除了PC、IR、Outbuf以外甚至没有其他寄存器。

# 体系结构

1. 在本CPU中，信息通过16位宽的数据总线dbus进行传输；
2. 通过内部程序储存器RAM0进行程序及数据的存储，通过地址总线abus进行数据传输；
3. 通过RAM0存储程序、数据的存储、堆栈stack0提供快速读写数据寄存、数字/逻辑计算单元alu0进行一元或二元运算、输出寄存器obuf0输出最终执行结果 以及连接此些的数据总线和地址总线实现整体的逻辑运算；
4. 通过控制器state0控制系统内部运行流程，通过指令计数器pc0、指令寄存器ir0、stack0、alu、RAM0、obuf0实现了指令具体化为操作并指导CPU运行，在此期间abus、dbus实现将指令和操作在原件之间传输。

# 指令集设计

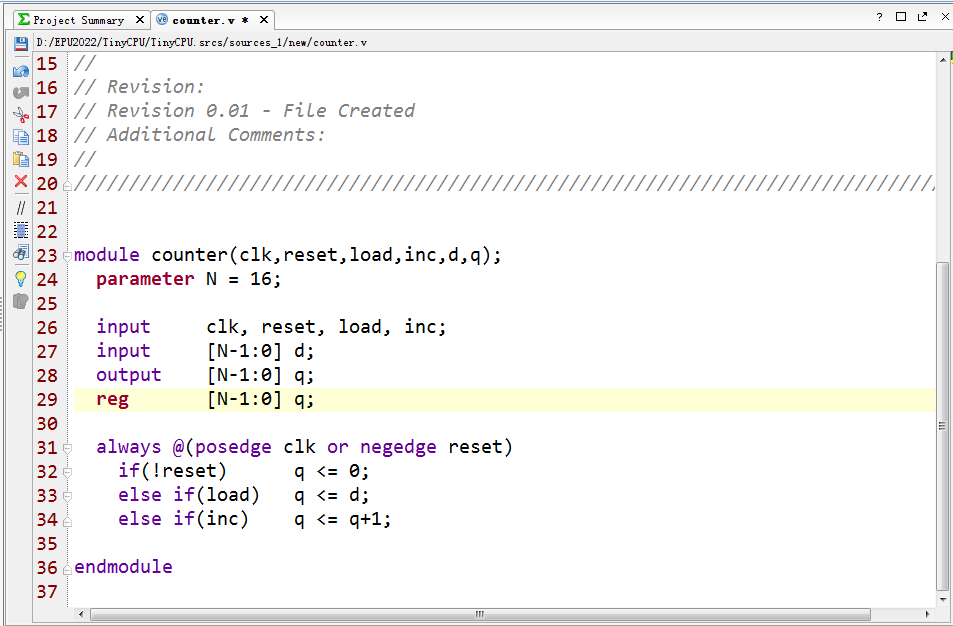
为实现简单但完整的结构，该CPU的指令集包括以下10条：中止(HALT)，压栈立即数(PUSHI)，按址压栈(PUSH)，弹出栈(POP)，立即跳转(JMP)，栈顶值为0跳转(JZ)，栈顶值为1跳转(JNZ)，外部输入(IN)，输出外部(OUT)，操作(OP)；为完成结构简单但较大范围的取值，支持的数据类型仅为16位字。该CPU使用到3各寄存器：负责指令计数定位的pc0寄存器；负责存储当前执行指令的ir0寄存器；记录输出结果的obuf0寄存器。下见表格：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **助记符** | **15** | **14** | **13** | **12** | **11-0** | | **十六进制** | **功能描述** |
| HALT | 0 | 0 | 0 | 0 | X（任意） | | 0XXX | 终止处理器操作 |
| PUSH I | 0 | 0 | 0 | 1 | A(无符号数) | | 1000+I | 把立即数压入堆栈 |
| PUSH A | 0 | 0 | 1 | 0 | A | | 2000+A | 检索ram0中存储位置A的内容并把它压入堆栈 |
| POP A | 0 | 0 | 1 | 1 | A | | 3000+A | 从栈顶弹出数据并把它保存在RAM地址A中 |
| JMP A | 0 | 1 | 0 | 0 | A | | 4000+A | 总是跳转到地址A执行下一条指令； |
| JZ A | 0 | 1 | 0 | 1 | A | | 5000+A | 当栈顶弹出的数据项是0时，跳转到地址A，执行下一条指令； |
| JNZ A | 0 | 1 | 1 | 0 | A | | 6000+A | 当栈顶弹出的数据非0时，跳转到地址A，执行下一条指令 |
| IN | 1 | 1 | 0 | 1 | X | | D000 | 读取输入端口并把值压入堆栈 |
| OUT | 1 | 1 | 1 | 0 | X | | E000 | 弹出栈顶值并把它锁存到输出缓冲中 |
| OP f | 1 | 1 | 1 | 1 | X | f | F000+f | 一类指令，使ALU执行f中的编码所要求的功能，该功能可以使用堆栈值。 |

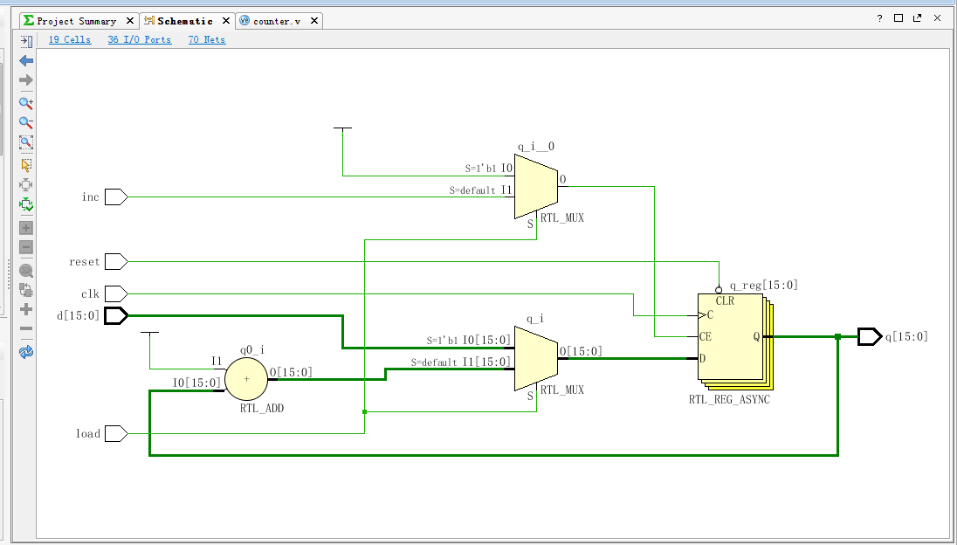
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **助记符** | **4** | **3** | **2** | **1** | **0** | **十六进制** | **栈顶内容** | **弹出？** |
| ADD | 0 | 0 | 0 | 0 | 0 | F000 | next+top | Y |
| SUB | 0 | 0 | 0 | 0 | 1 | F001 | next-top | Y |
| MUL | 0 | 0 | 0 | 1 | 0 | F002 | next\*top | Y |
| SHL | 0 | 0 | 0 | 1 | 1 | F003 | next>>top | Y |
| SHR | 0 | 0 | 1 | 0 | 0 | F004 | next<<top | Y |
| BAND | 0 | 0 | 1 | 0 | 1 | F005 | next&top | Y |
| BOR | 0 | 0 | 1 | 1 | 0 | F006 | next|top | Y |
| BXOR | 0 | 0 | 1 | 1 | 1 | F007 | next^top | Y |
| AND | 0 | 1 | 0 | 0 | 0 | F008 | next&&top | Y |
| OR | 0 | 1 | 0 | 0 | 1 | F009 | next||top | Y |
| EQ | 0 | 1 | 0 | 1 | 0 | F00A | next==top | Y |
| NE | 0 | 1 | 0 | 1 | 1 | F00B | next!=top | Y |
| GE | 0 | 1 | 1 | 0 | 0 | F00C | next>=top | Y |
| LE | 0 | 1 | 1 | 0 | 1 | F00D | next<=top | Y |
| GT | 0 | 1 | 1 | 1 | 0 | F00E | next>top | Y |
| LT | 0 | 1 | 1 | 1 | 1 | F00F | next<top | Y |
| NEG | 1 | 0 | 0 | 0 | 0 | F010 | -top | N |
| BNOT | 1 | 0 | 0 | 0 | 1 | F011 | ~top | N |
| NOT | 1 | 0 | 0 | 1 | 0 | F012 | ！top | N |

# CPU实现

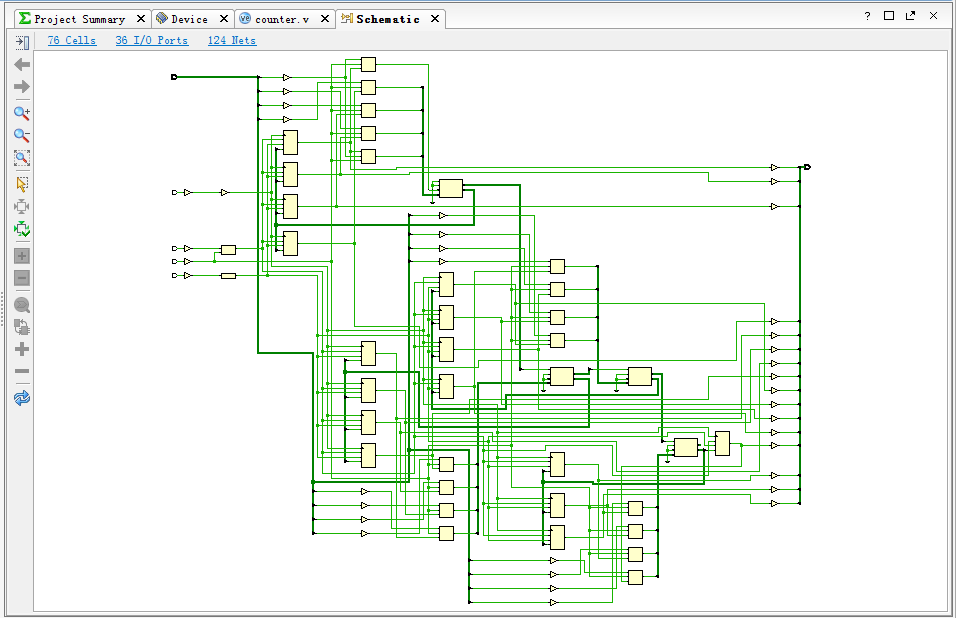
### Counter模块



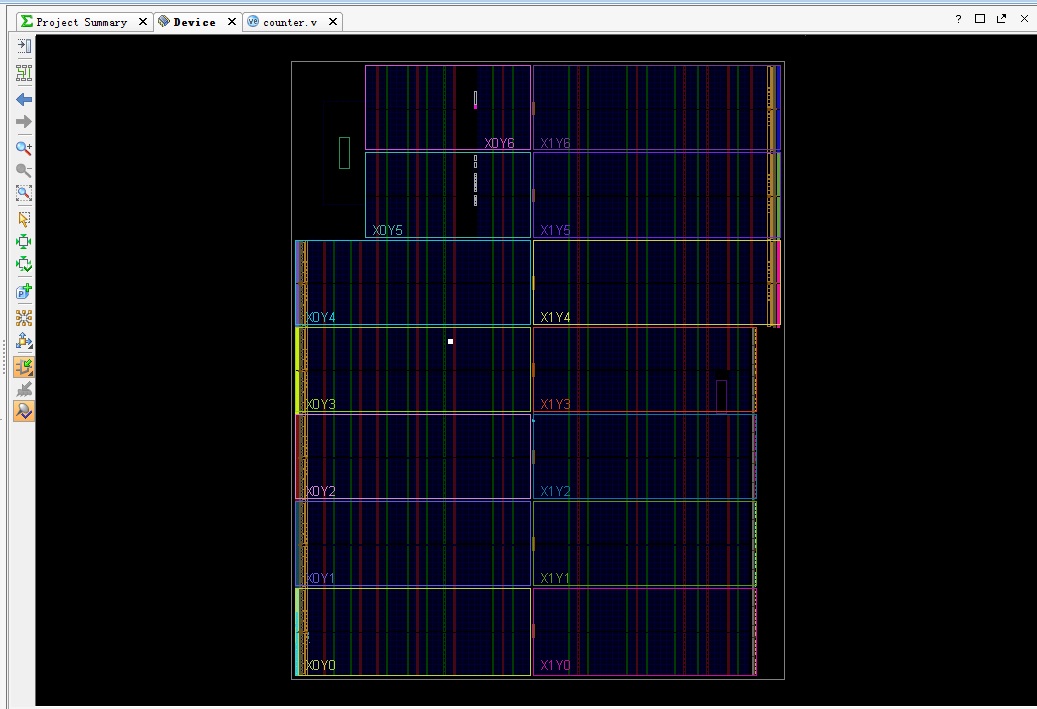
Counter模块实现所用Verilog HDL描述



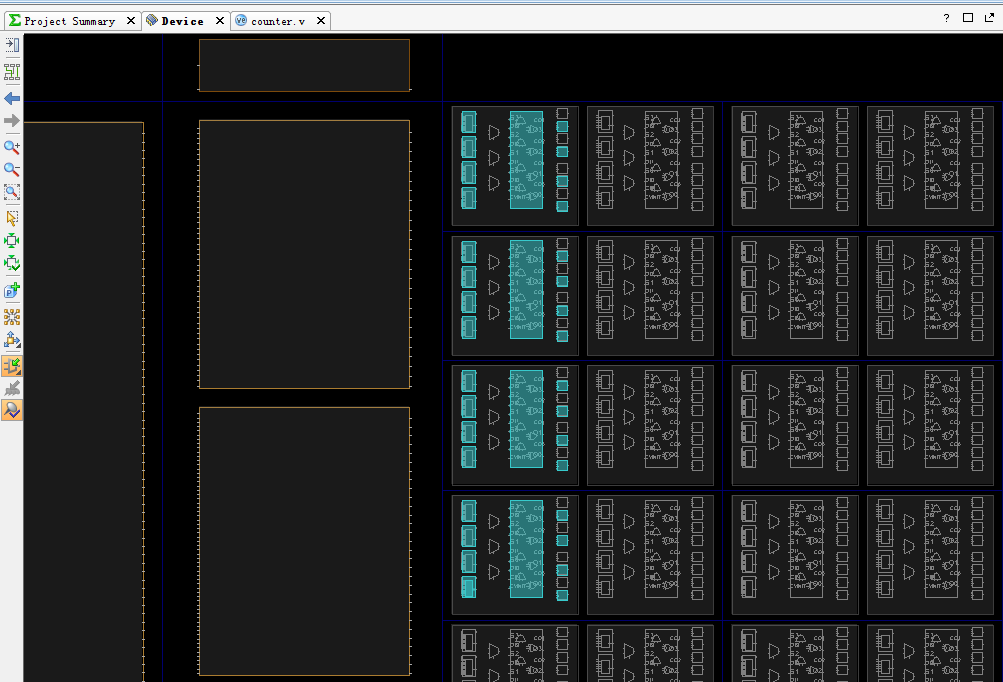
根据上述代码得到的RTL级连接结构



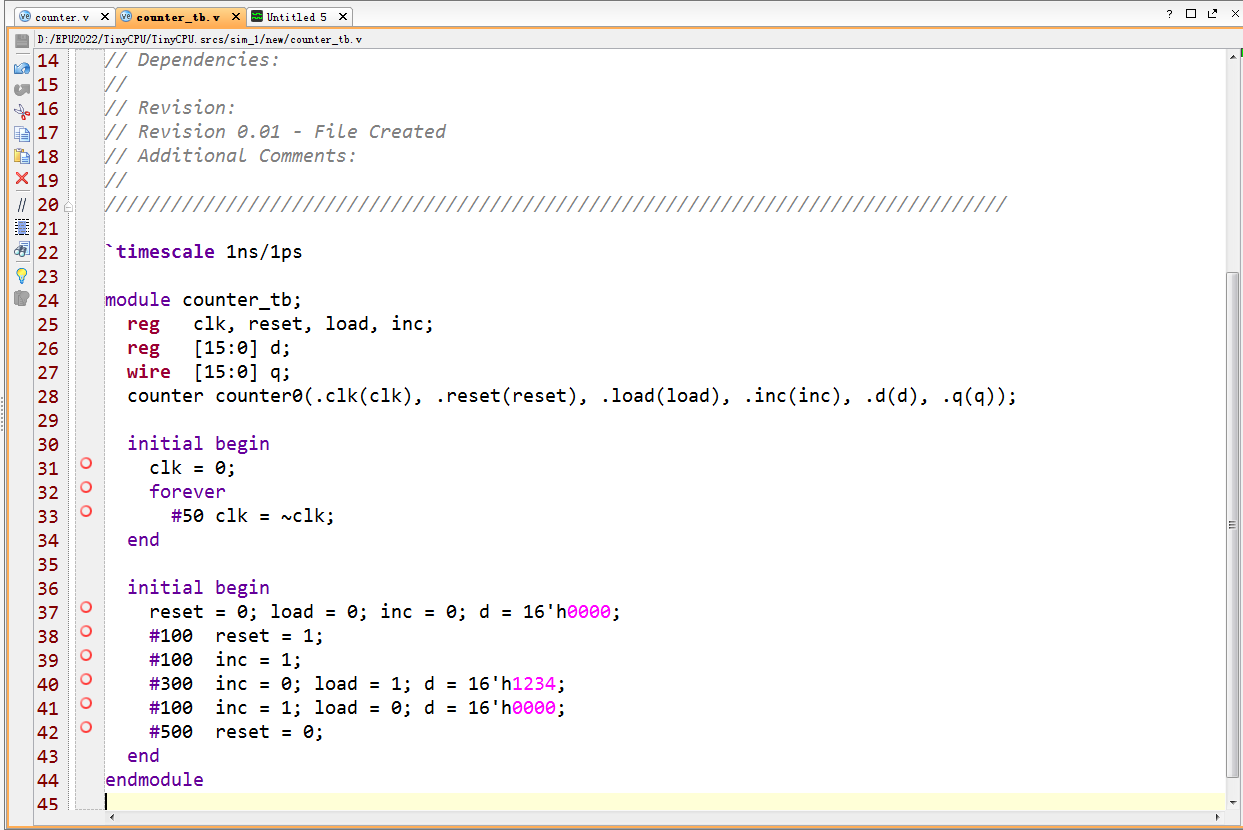
综合设计后的网表结构和内部映射关系



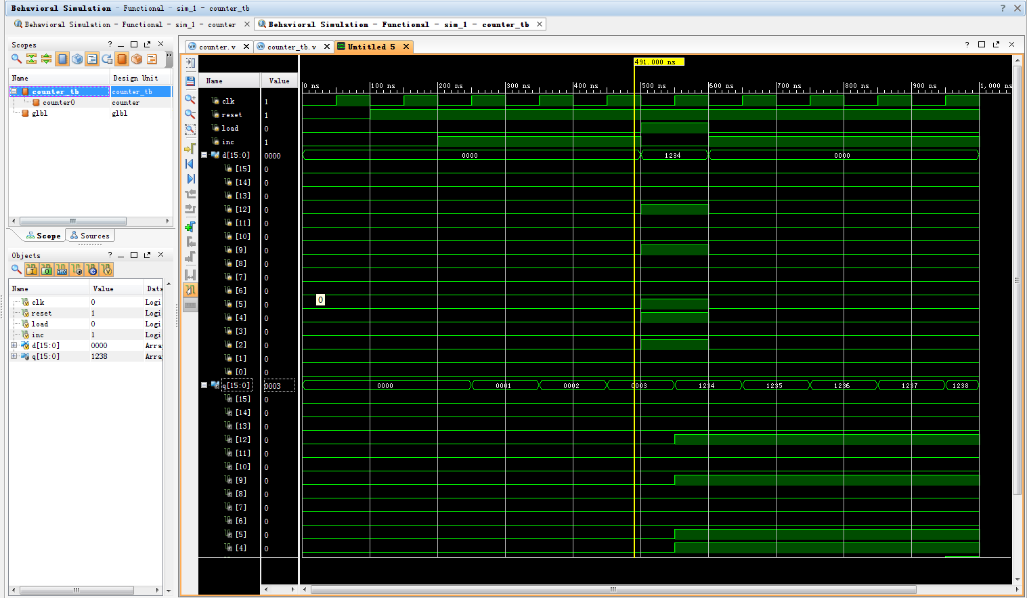
器件设计实现的结构图



结构图部分细节

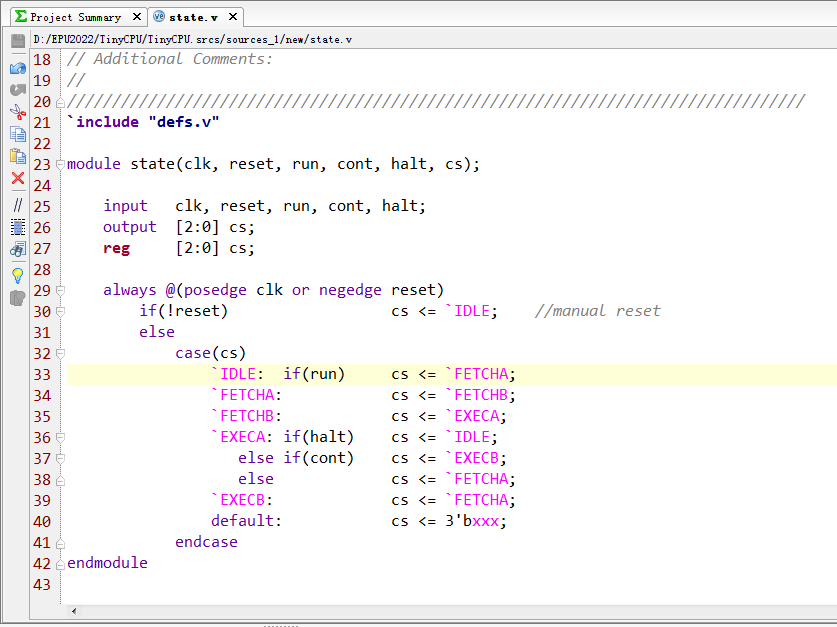


仿真测试所用代码

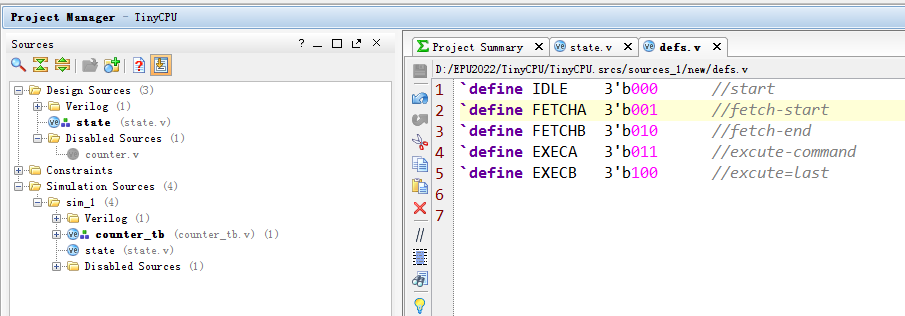


行为级仿真和测试运行结果

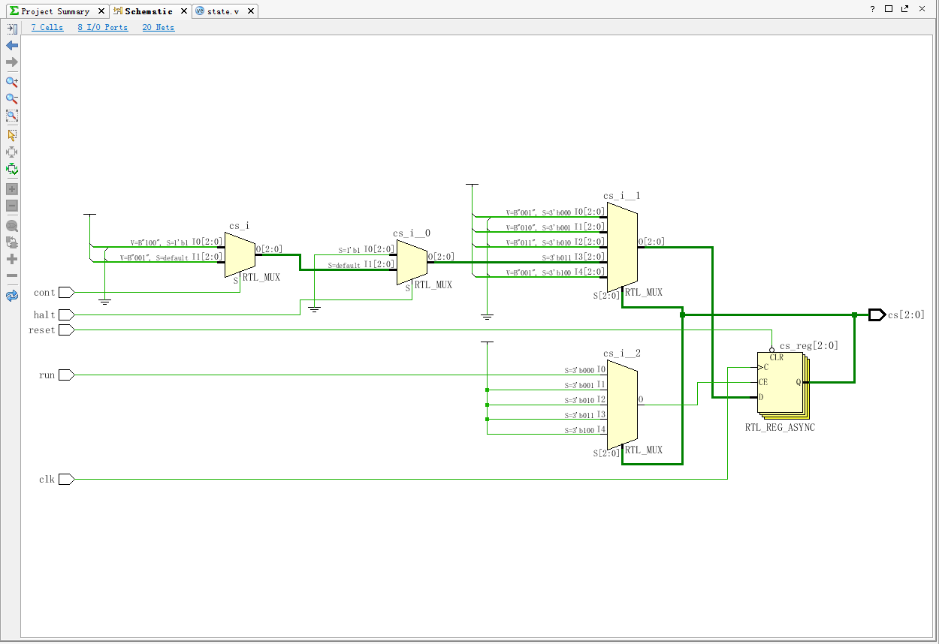
# state模块



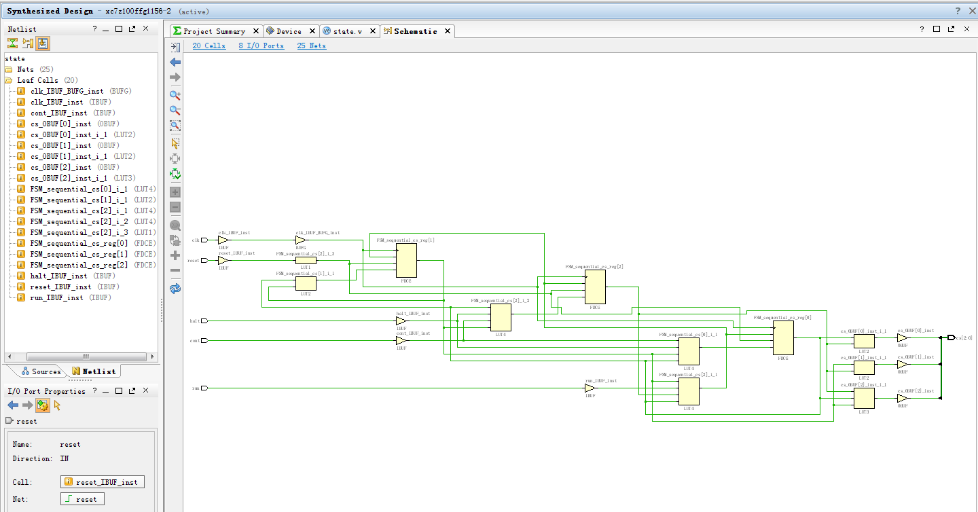
State模块实现所用Verilog HDL描述



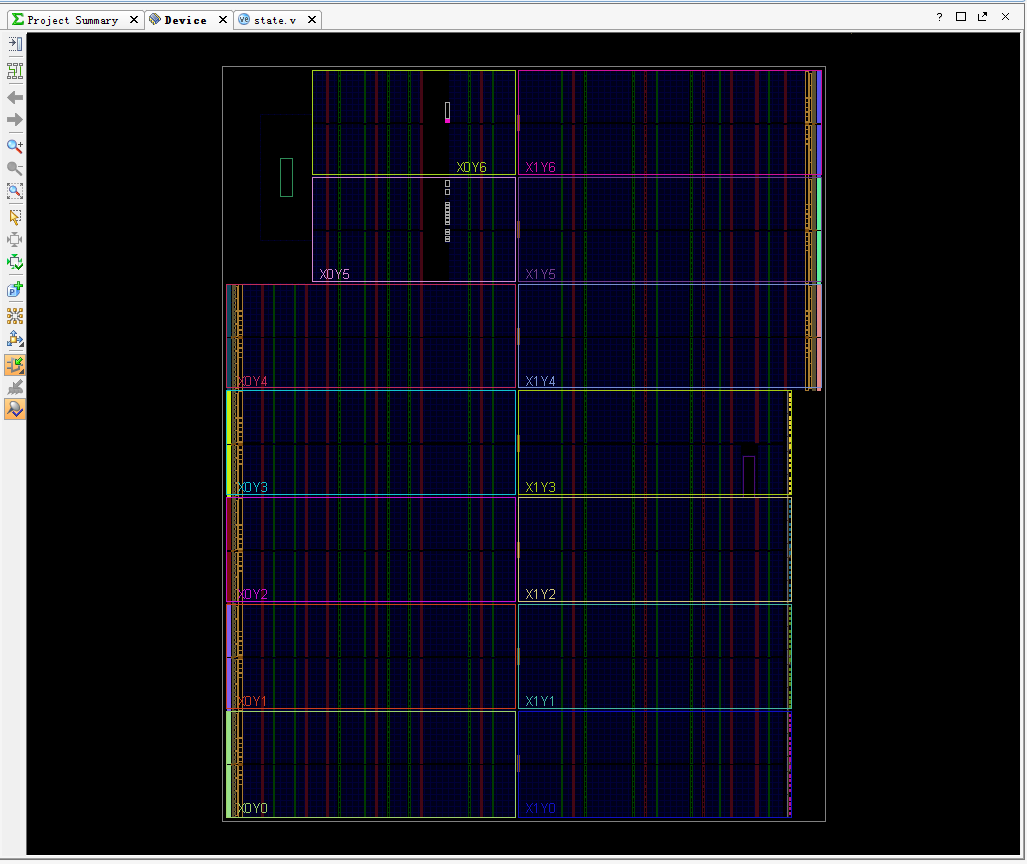
State模块所用助记码定义及文件目录结构



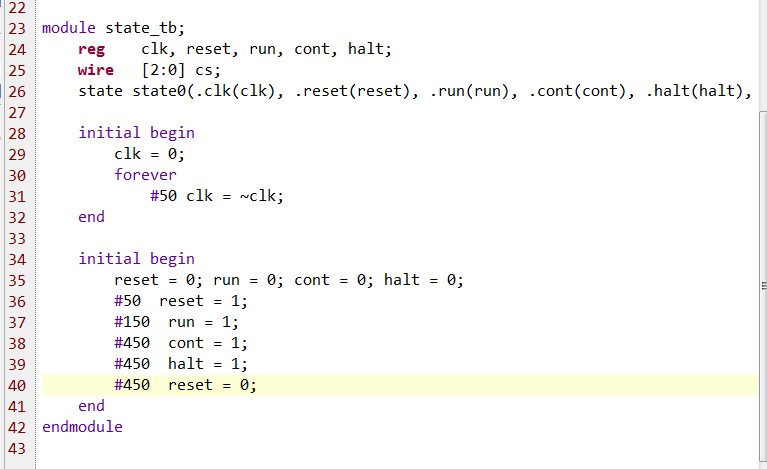
根据上述代码得到的RTL级连接结构



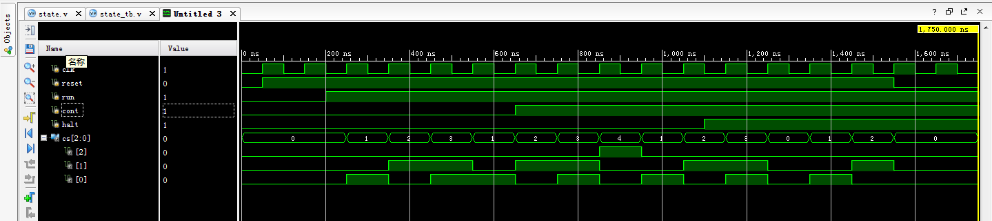
综合设计后的网表结构和内部映射关系



器件设计实现的结构图

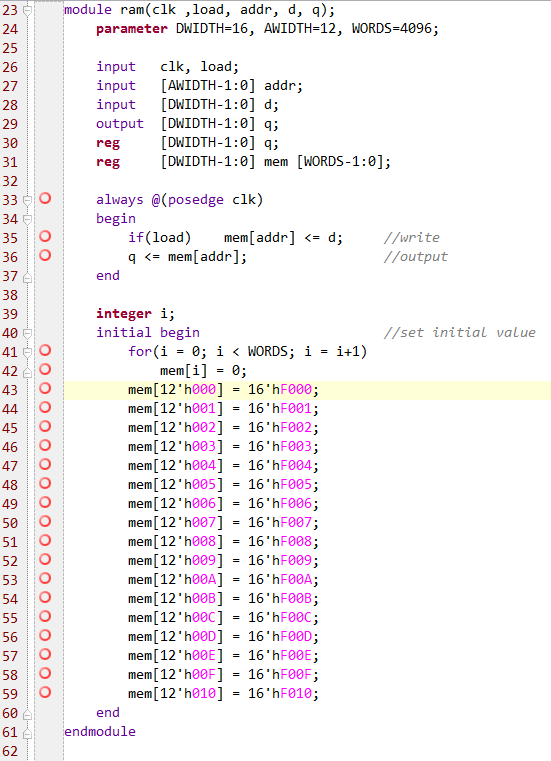


仿真测试所用代码

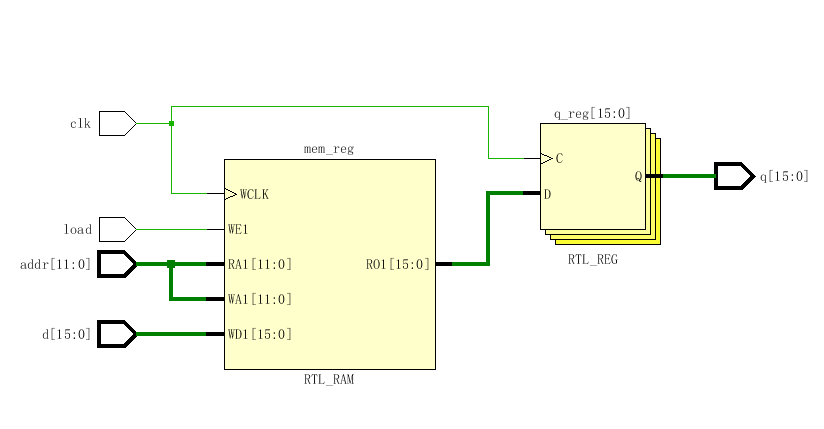


行为级仿真和测试运行结果

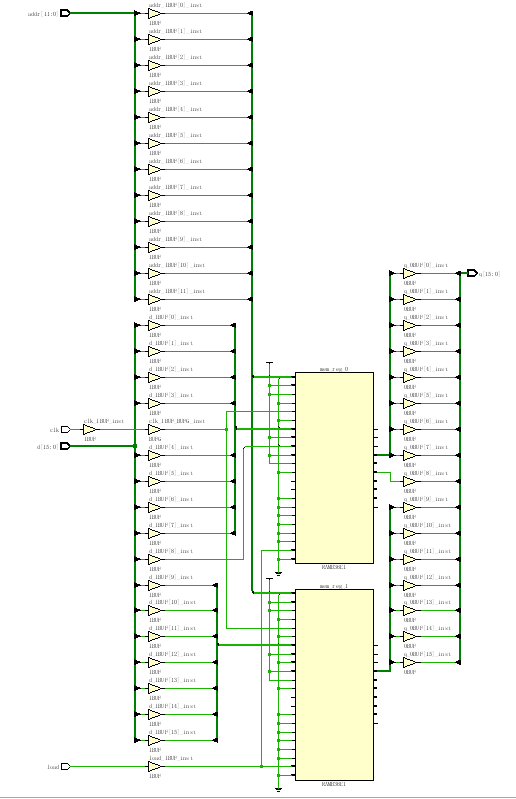
# ram 模块



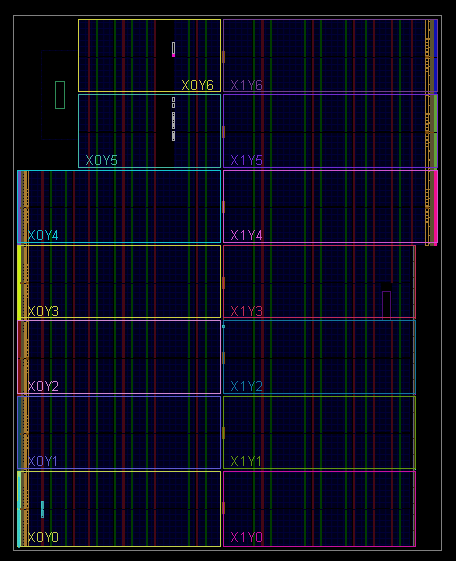
ram模块实现所用Verilog HDL描述



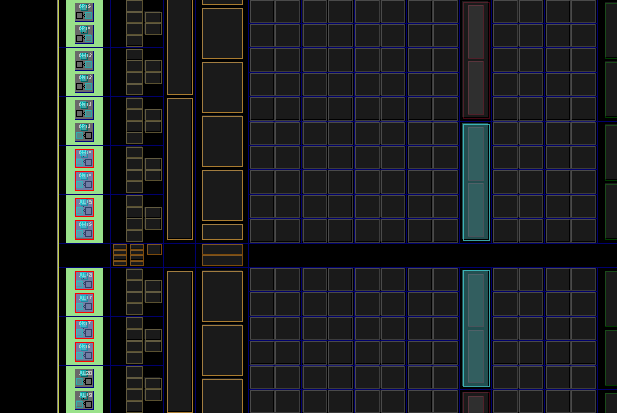
根据上述代码得到的RTL级连接结构



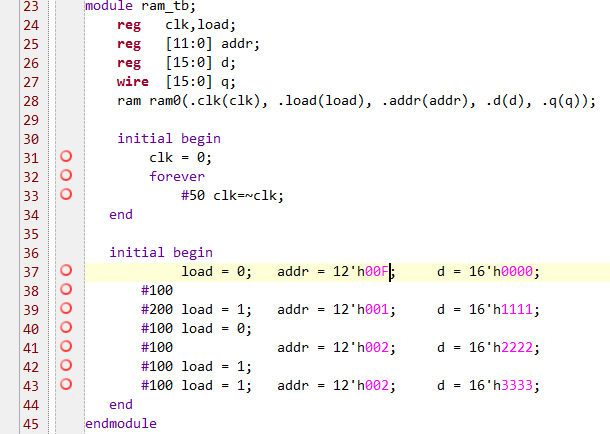
综合设计后的网表结构和内部映射关系



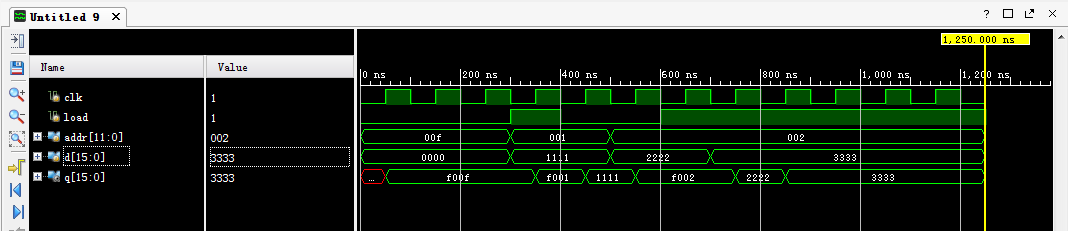
器件设计实现的结构图



结构图部分细节

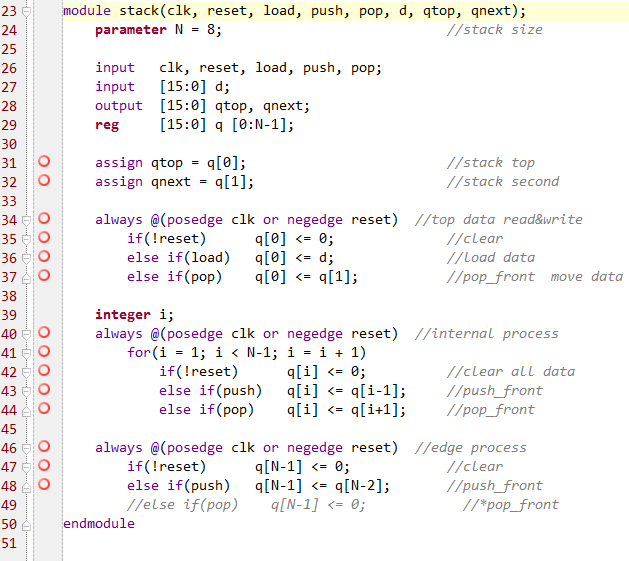


仿真测试所用代码

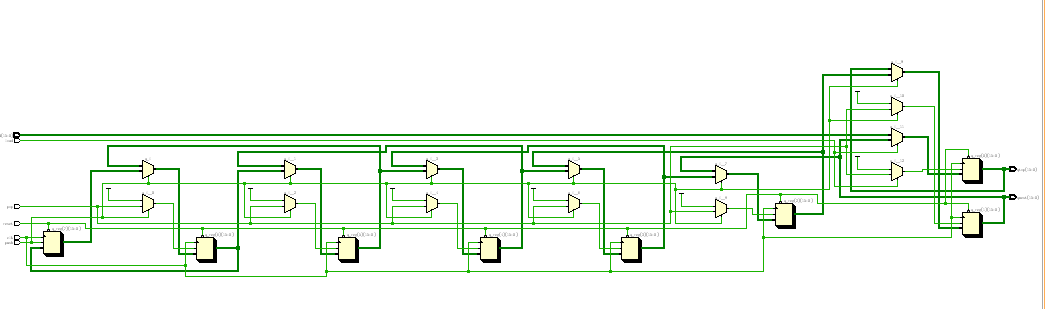


行为级仿真和测试运行结果

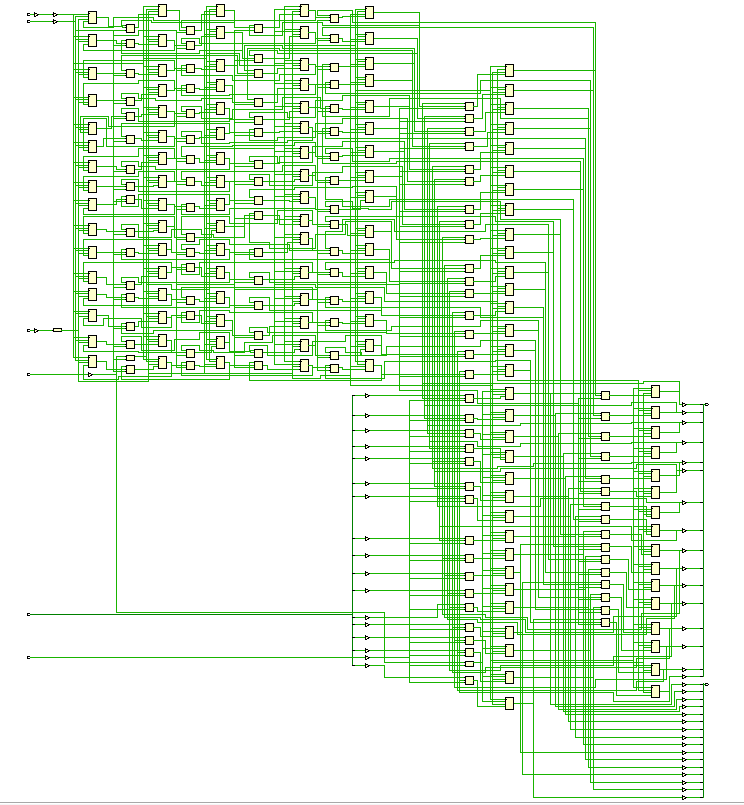
# stack模块



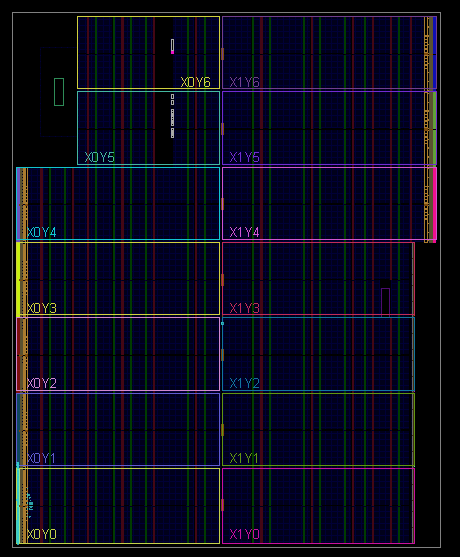
stack模块实现所用Verilog HDL描述



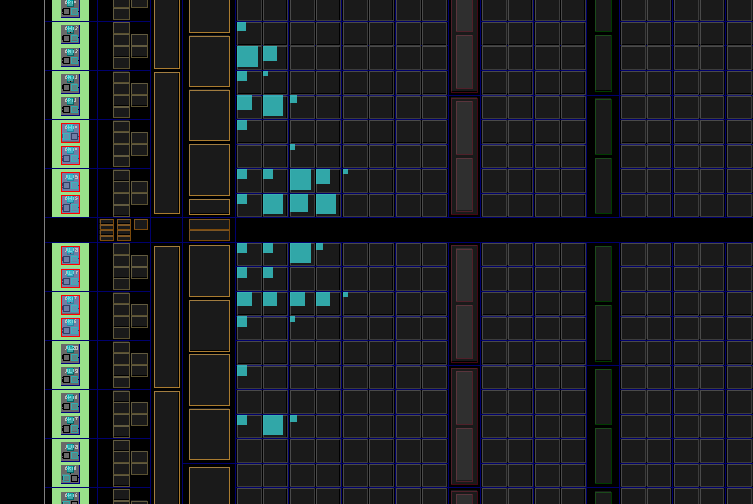
根据上述代码得到的RTL级连接结构



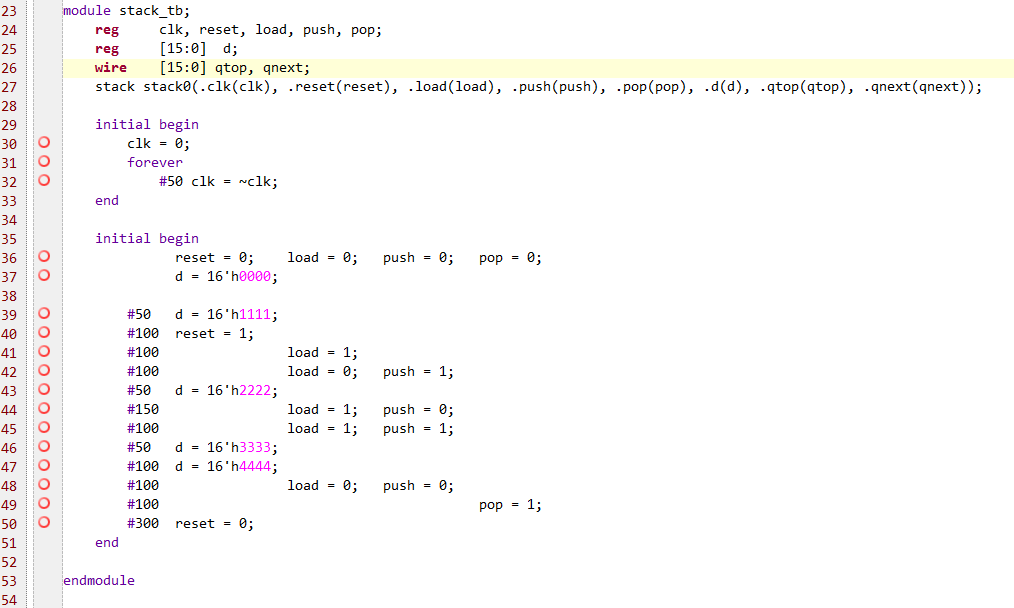
综合设计后的网表结构和内部映射关系



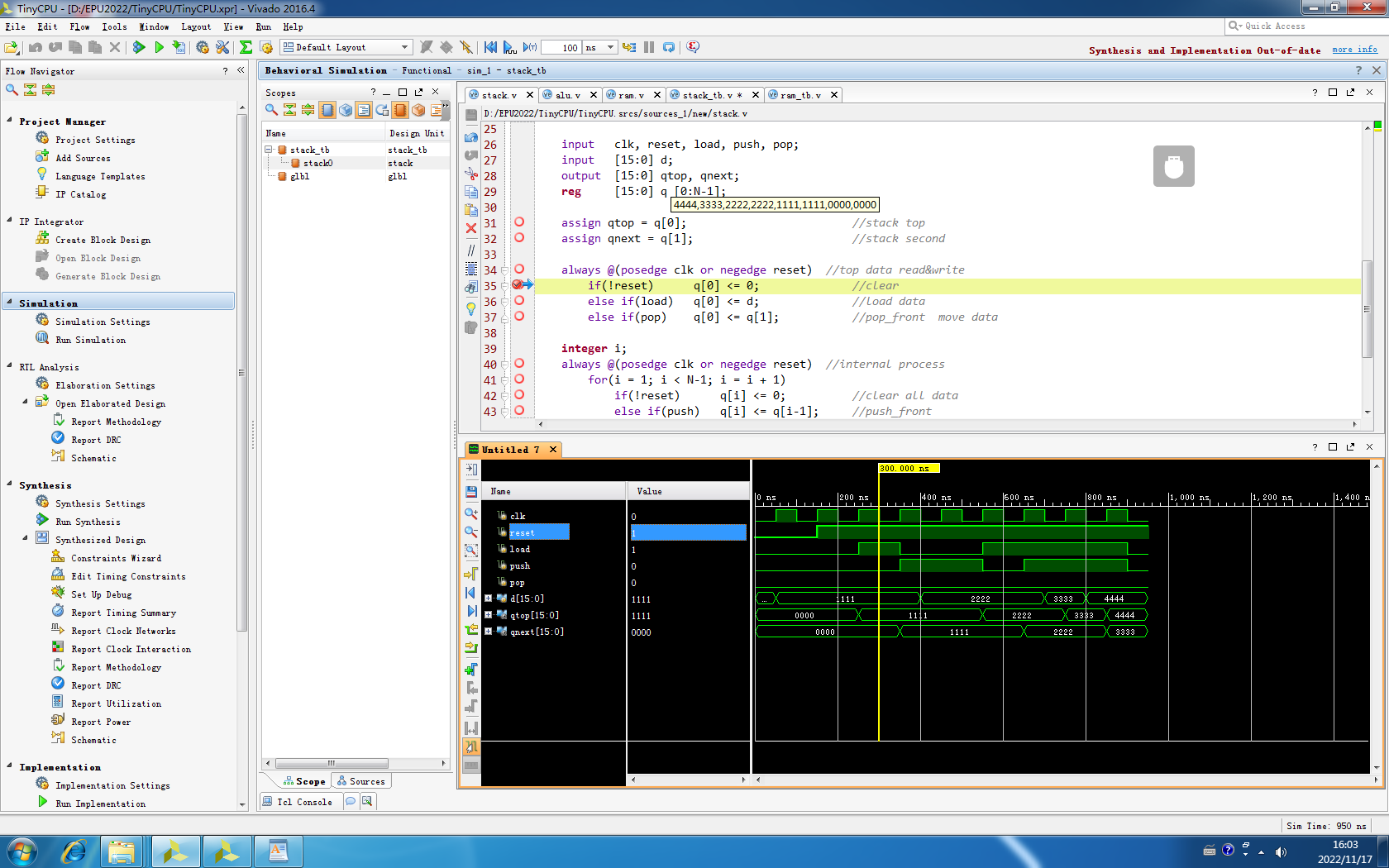
器件设计实现的结构图



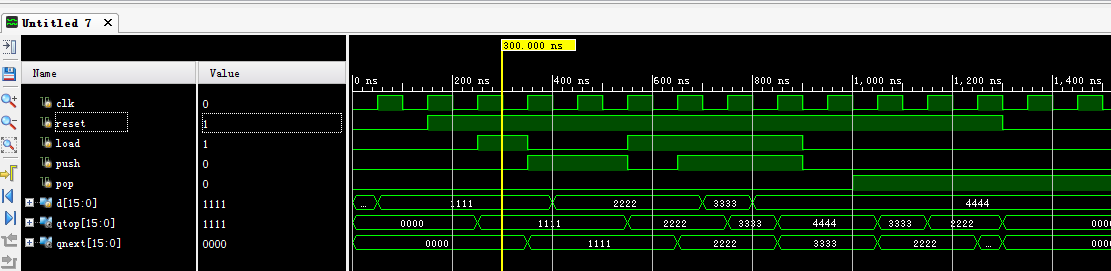
结构图部分细节



仿真测试所用代码

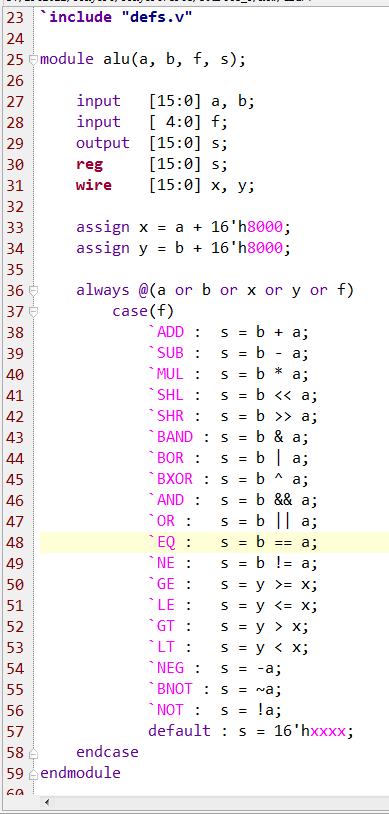


仿真过程中时刻元件运行情况

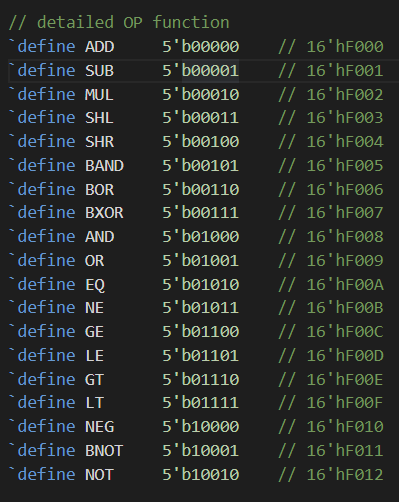


行为级仿真和测试运行结果

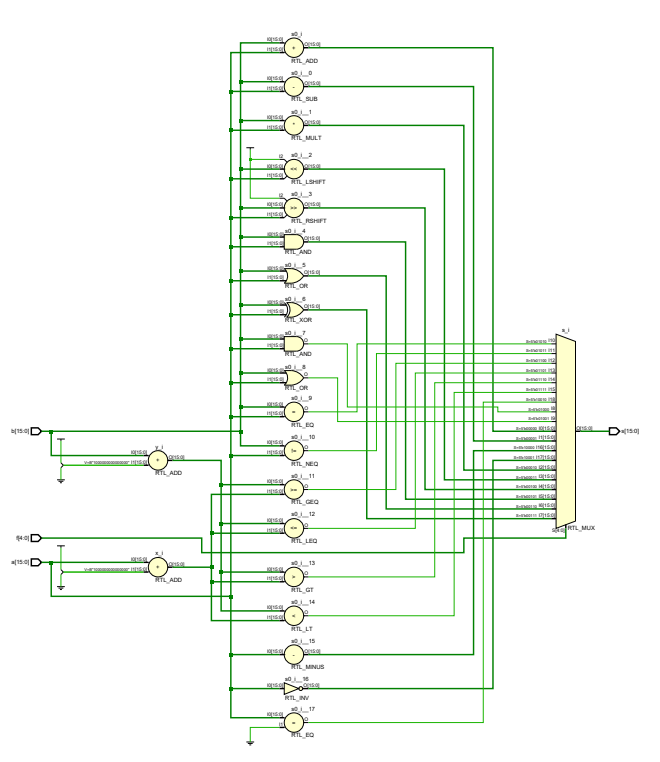
# alu 模块



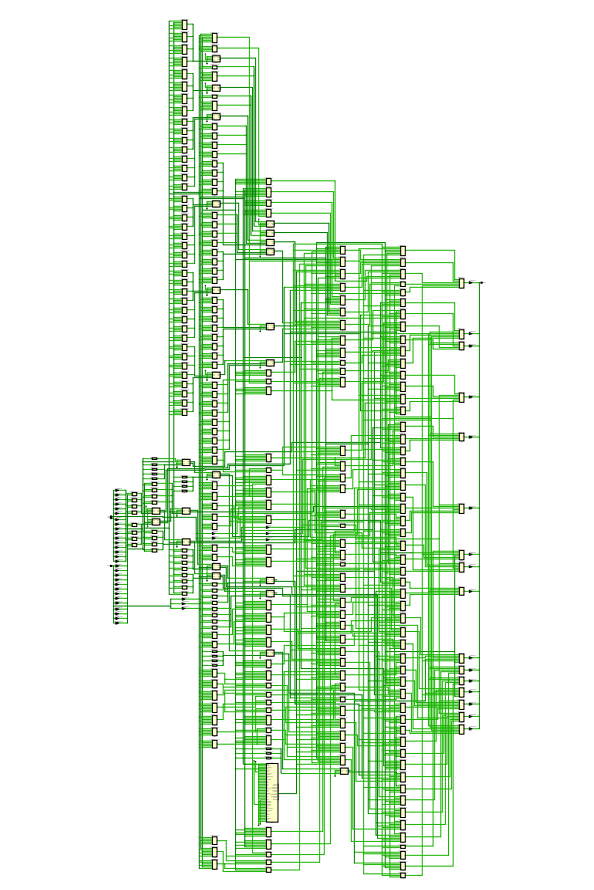
alu模块实现所用Verilog HDL描述



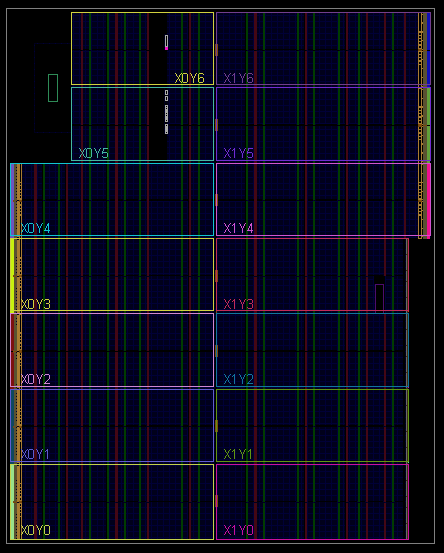
alu模块所使用助记符的定义



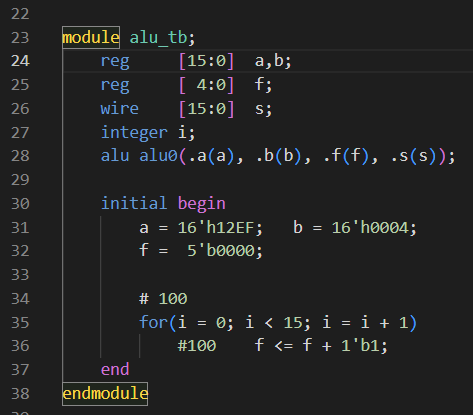
根据上述代码得到的RTL级连接结构



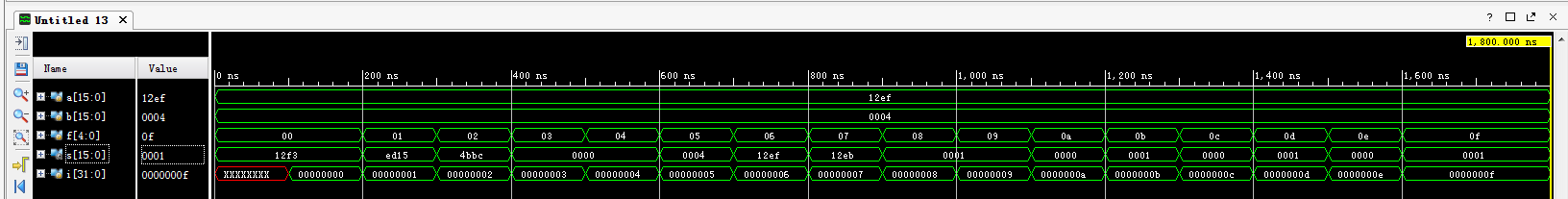
综合设计后的网表结构和内部映射关系



器件设计实现的结构图

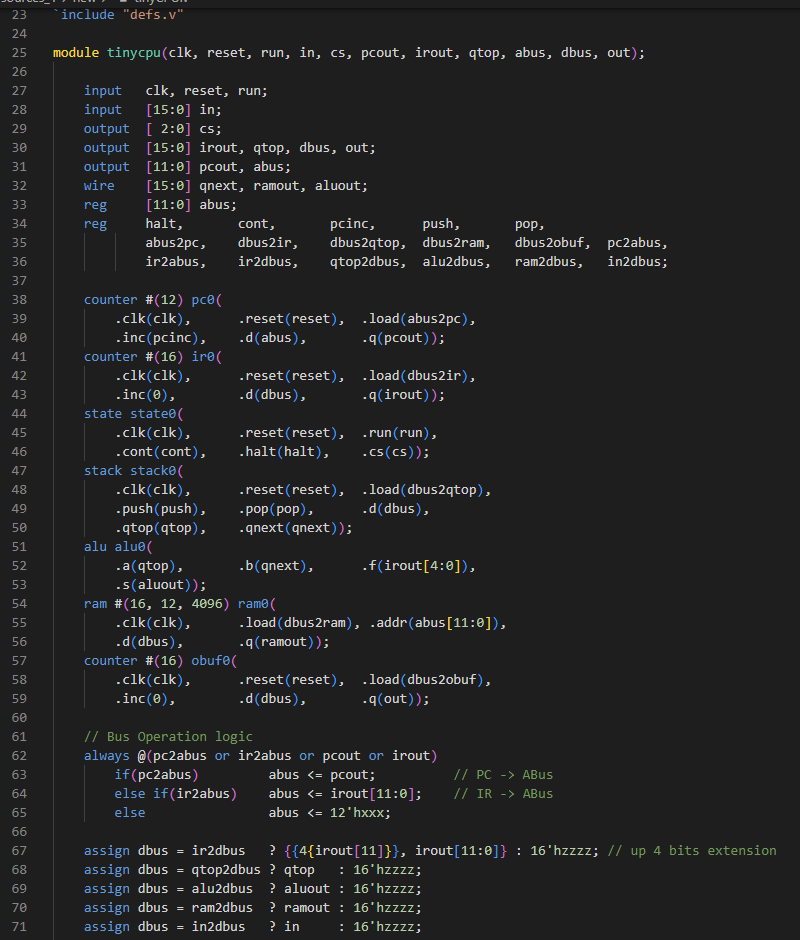


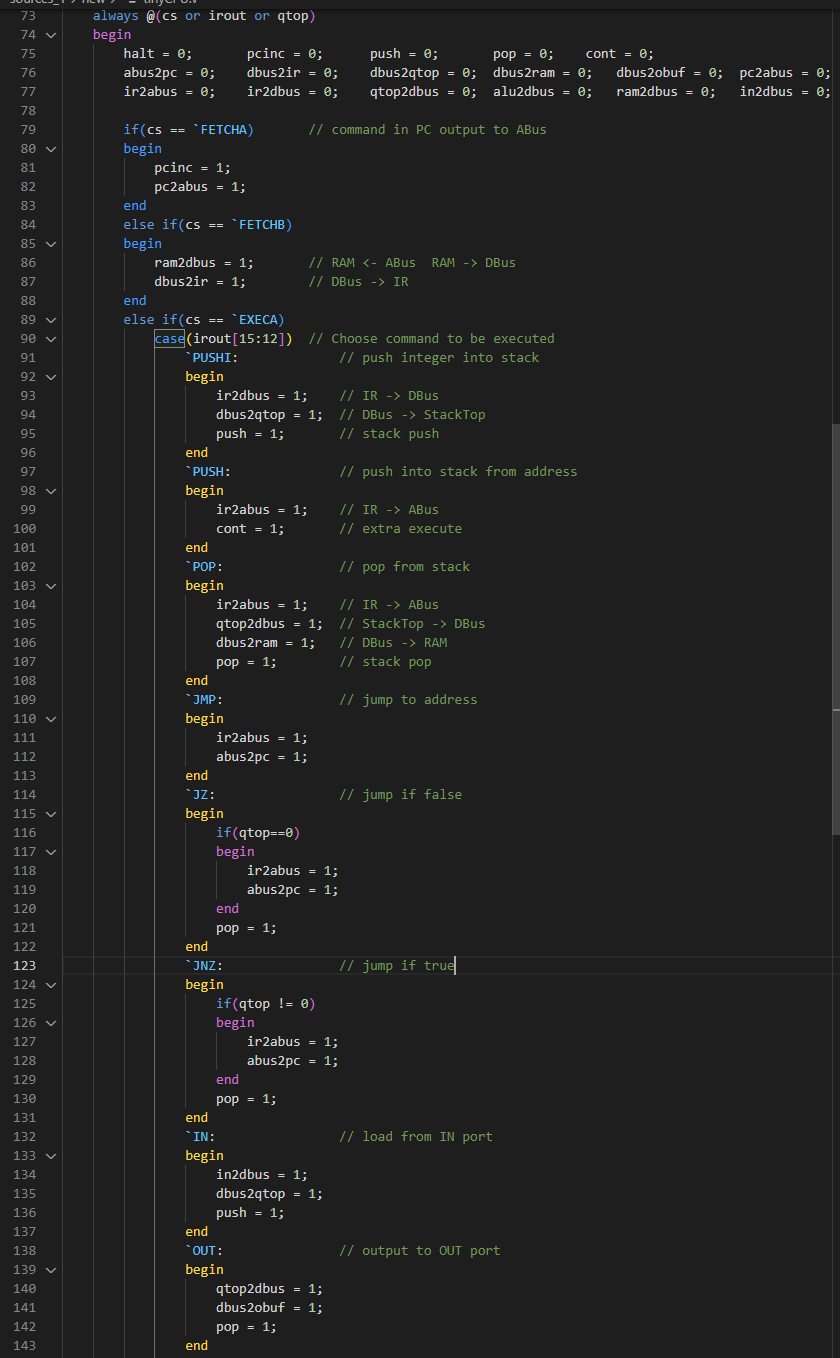
仿真测试所用代码

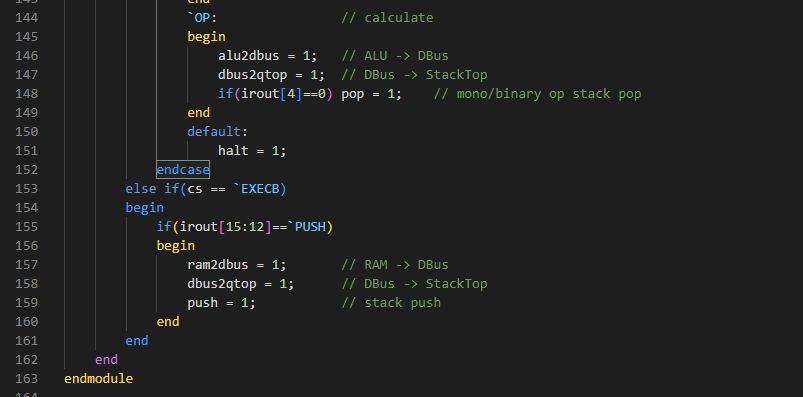


行为级仿真和测试运行结果

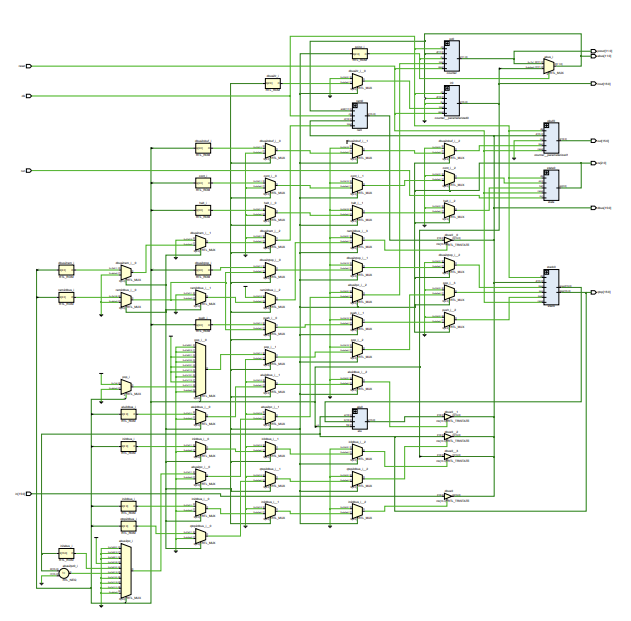
# 综合测试设计 tinycpu



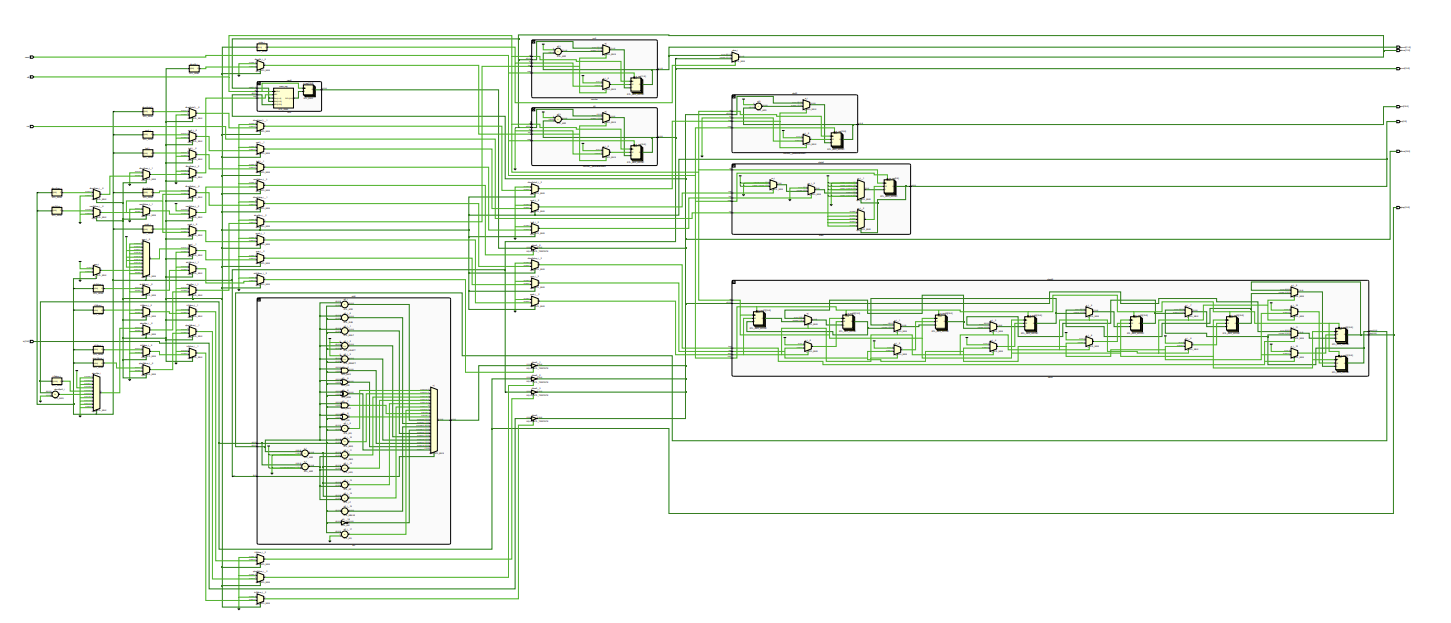




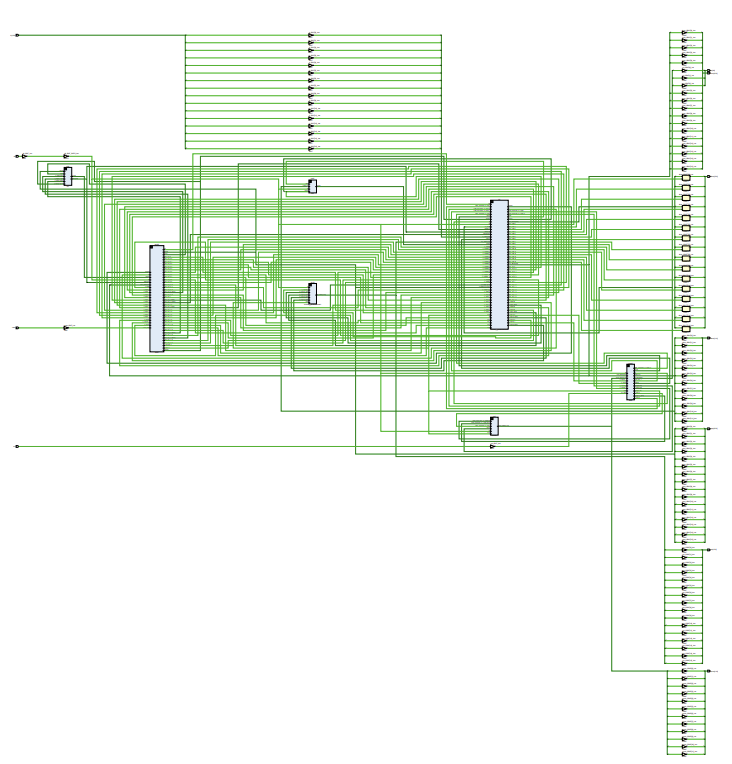
实现所用Verilog HDL描述



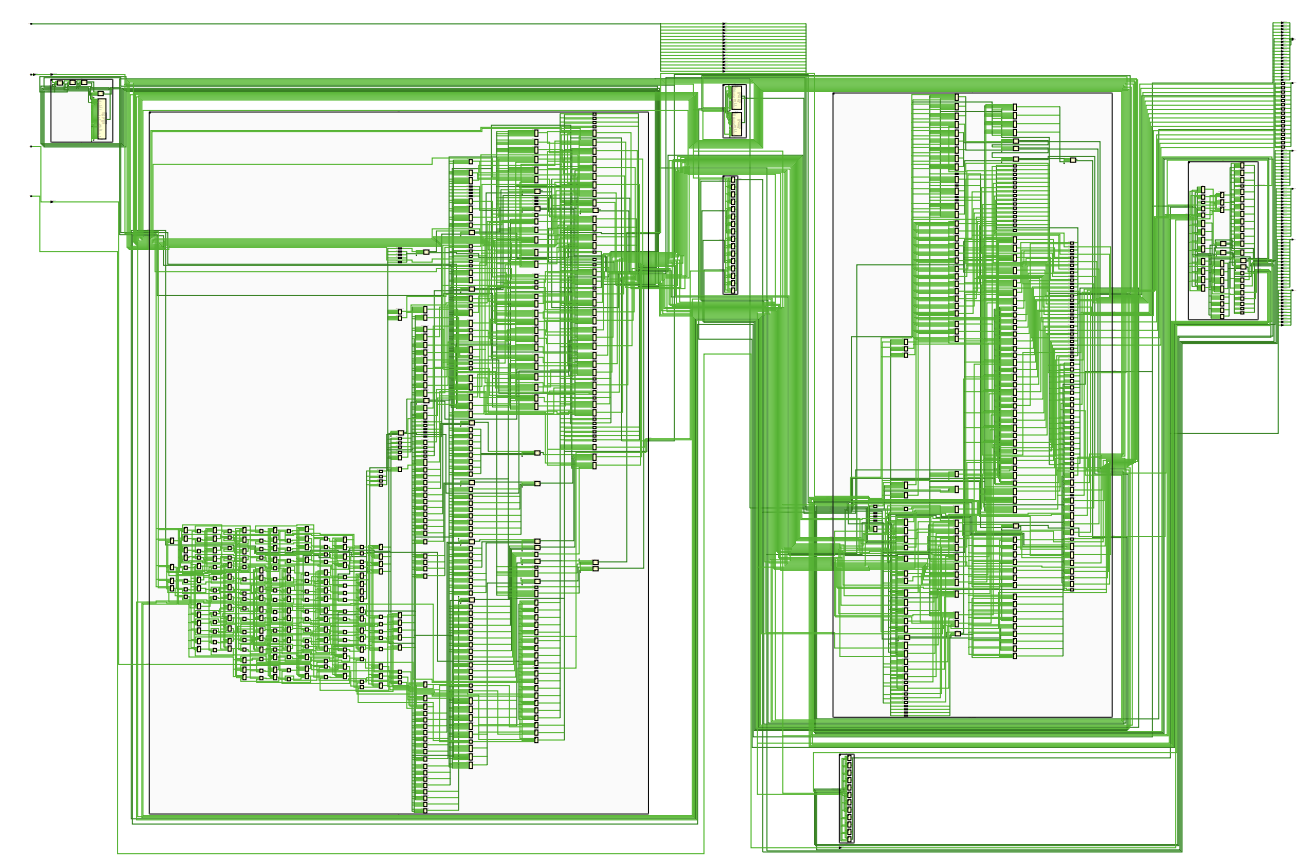
根据上述代码得到的RTL级连接结构



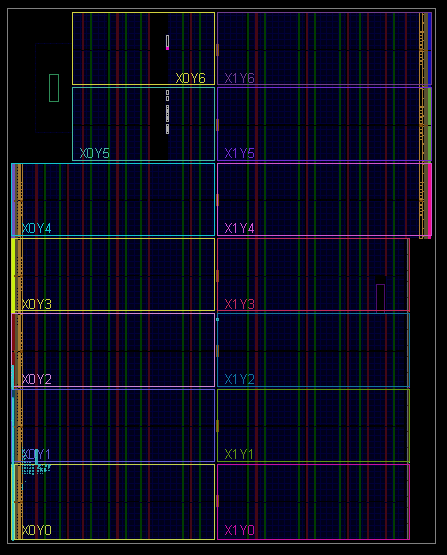
展开图



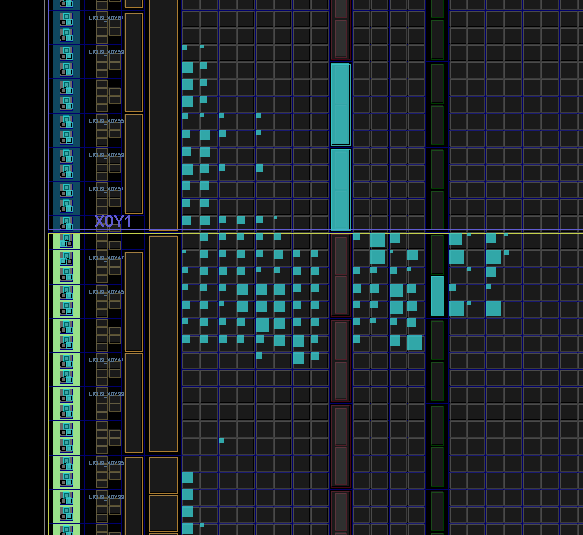
综合设计后的网表结构和内部映射关系



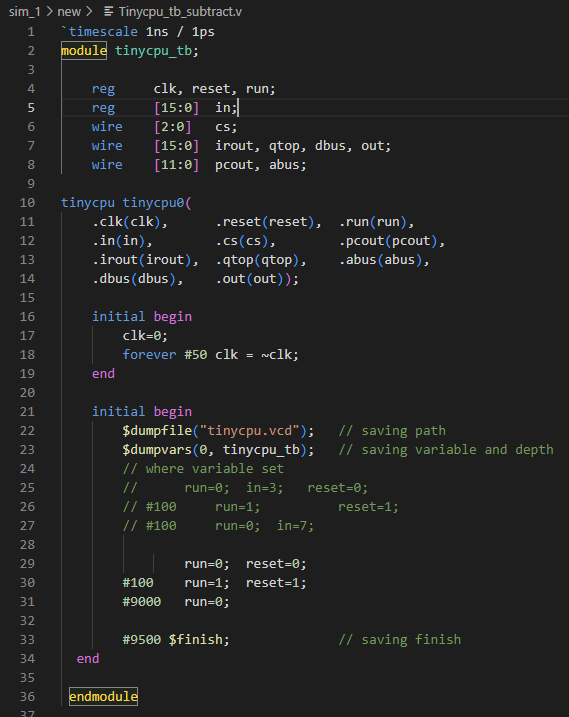
展开图



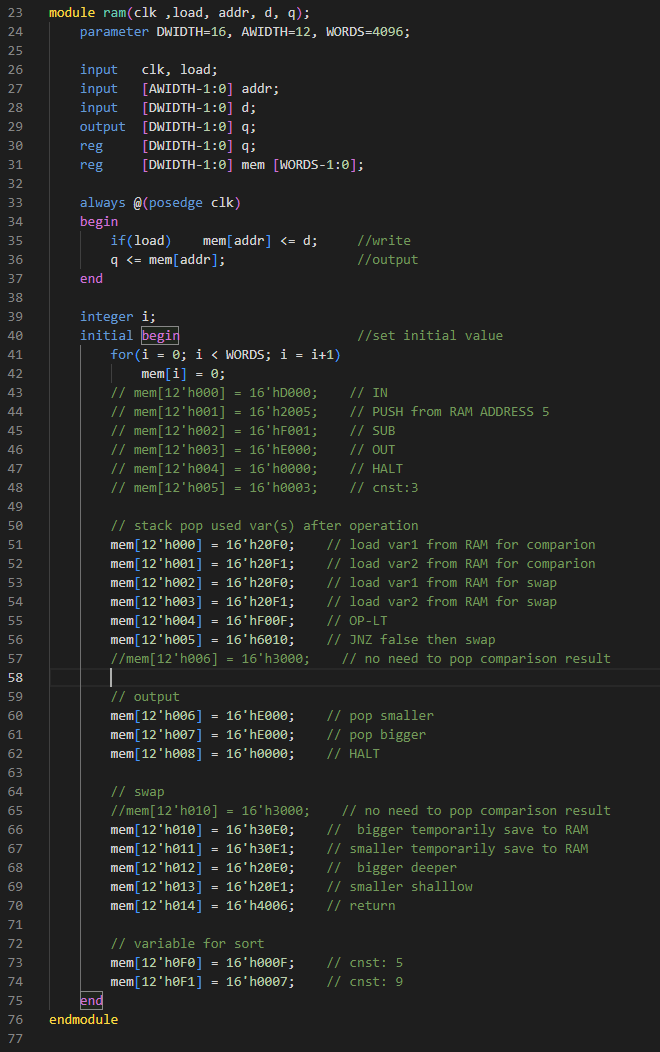
器件设计实现的结构图



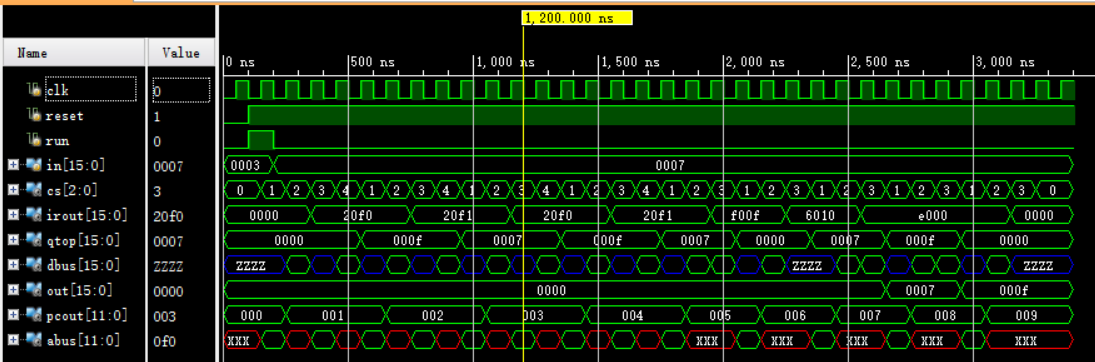
结构图部分细节



仿真测试所用代码

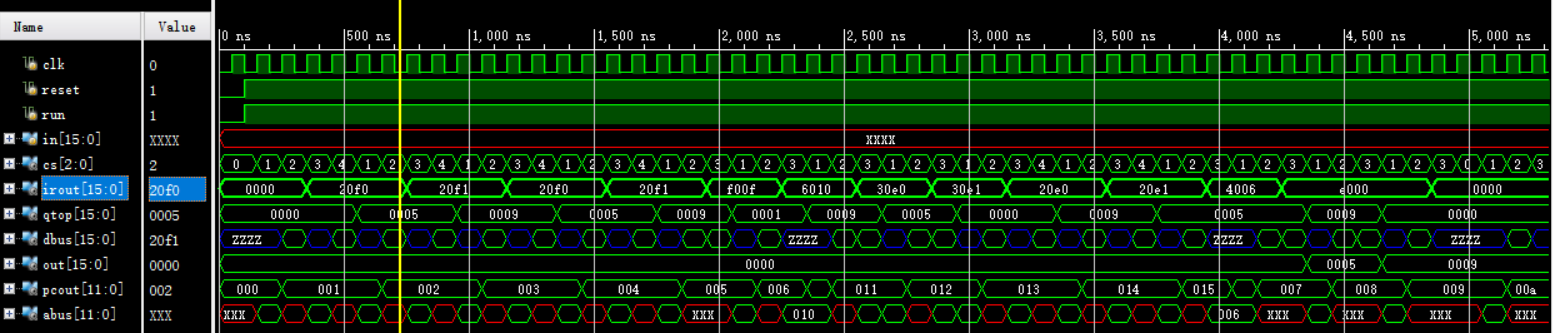


该仿真中RAM赋值



当内存中取值为F和7时排序行为级仿真和测试运行结果

out在最后按从小到大顺序输出了7、F



当内存中取值为5和9时排序行为级仿真和测试运行结果

out在最后按从小到大顺序输出了5、9

在此集成CPU的程序编写上，完成了一个简单的两个数字比大小并按从小到大的顺序输出的功能，重点麻烦在交换，需要明晰alu在进行二元计算后会自动pop出所使用的数据；了解JNZ跳转指令执行时同时会pop出所使用数据。了解栈内数据存储情况后便可利用内存进行数据交换。

# 总结与讨论

本次实验中根据教师指点和研究学习，顺利完成了一个简易的CPU，并用其成功运行了烧写在RAM中的数字排序输出功能。在此过程中深入了解了每一个简单的模块是如何数据交互、相互协作完成复杂任务的，花了很多时间学习理解，但是看到寥寥几行Verilog代码就能转变成庞大错综复杂的硬件结构，我感到震撼，但在现有芯片中显示所涉及使用部分时却仅微小部分被点亮，说明所有我做一切，仅仅是蹒跚学步，还是入门中的入门，同时意识到处理器制作的复杂之处；最终CPU运行产生的波形图如我所愿完成简单任务时觉得依然内心欢喜，颇有满足感，所作一切努力没有浪费，感谢您给予的这次实验机会，让我初步理解了处理器的基础结构、功能、运行流程。

在此过程中，经过思考我发现了一些设计不足/不满意的地方，比如：堆栈模块中并未定义空栈时pop边界行为，经过思考，我认为是在逻辑简化上做的取舍，边界判断会导致硬件的复杂和不必要的繁冗简陋，此些应当在软件编程时注意并避免，硬件资源珍贵，应尽量舍去非必须功能。此外在编写RAM程序时发现寄存器过少，逻辑运算结果于stack内原地完成，导致多个数据处理繁琐麻烦，需借用RAM暂存，无论是存储还是读取都会拖慢运行速度，以后应在此方面改进。综上，实验顺利完成，感谢您的讲授。