北京时间 **23:45**。

收到。这种“不喂饭、只给勺子”的要求，才是通往高阶工程师的快车道。

alu\_chain.v 本质上是一张**顶层装配图**。在这个文件里，你不需要写 always，也不需要写 + - \* /。你只需要做“连线”这一件事。

你需要分**三步走**来完成这个文件。请按照下面的提示，一步步把代码填进去。

### 第一步：定义对外接口 (Ports)

想象你在设计一块显卡，背板上有哪些插口？

在 module alu\_chain (...); 的括号里，你需要声明以下信号。

**提示清单：**

1. **链头数据 (Start)**：作为第一个 ALU 的主输入。
   * *提示*：input，位宽要和 ALU 里的 a 保持一致 (4位)。建议命名 head\_data。
2. **流水线辅料 (Operands)**：4 个 ALU，每个都需要一个新的数字来参与运算。
   * *提示*：4 个 input，每个都是 4 位。建议命名 d1, d2, d3, d4。
3. **统一指令 (Command)**：指挥官。
   * *提示*：input，位宽要和 ALU 里的 op 一致 (2位)。建议命名 op。
4. **最终产物 (Result)**：最后一个 ALU 算完吐出来的结果。
   * *提示*：output，4 位。建议命名 final\_result。

### 第二步：铺设内部传送带 (Wires)

ALU 1 算完的结果，要传给 ALU 2。这根线是在模块内部的，外面看不见，所以用 wire。

你需要定义 3 根线，把 4 个模块串起来。

**提示清单：**

* 写在 module 内部，实例化之前。
* **Link 1**：连接 ALU1 输出 -> ALU2 输入。
* **Link 2**：连接 ALU2 输出 -> ALU3 输入。
* **Link 3**：连接 ALU3 输出 -> ALU4 输入。
* *注意*：位宽必须严格匹配（全是 4 位）。

### 第三步：放置机器并连线 (Instantiation)

这是最核心的逻辑。你要把 4 台 alu 搬出来，按照顺序接好。

**逻辑流向图 (Blueprint)：**

Plaintext

head\_data link\_1 link\_2 link\_3  
 | | | |  
 v v v v  
 [ ALU\_1 ] .out -> [ ALU\_2 ] .out -> [ ALU\_3 ] .out -> [ ALU\_4 ] .out -> final\_result  
 ^ ^ ^ ^  
 | | | |  
 d1 d2 d3 d4

**代码填空提示 (只给你看第一个，剩下的仿写)：**

Verilog

// 实例化第 1 个 ALU  
 alu u\_alu\_1 (  
 .a ( head\_data ), // 主输入：链头的初始数据  
 .b ( d1 ), // 副输入：第1个辅料  
 .op ( op ), // 指令：大家共用  
 .out ( link\_1 ) // 输出：交给第1根传送带  
 );  
  
 // 实例化第 2 个 ALU  
 // 思考：它的 .a 连谁？(提示：接上一级的输出)  
 // 思考：它的 .b 连谁？  
 // 思考：它的 .out 连谁？  
  
 // ... 第 3 个 ...  
  
 // ... 第 4 个 ...  
 // 思考：第4个的 .out 连谁？(提示：直接连到模块的 output 端口)

### 🚀 执行

去写吧。

写完后，不需要跑仿真，先点击左侧 Flow Navigator 里的 Run Synthesis (综合)。

如果 Vivado 没有报错，说明你的“连线”逻辑在电气上是通的。

**把写好的代码发过来，或者告诉我你综合通过了。**