1、 不了解模块的实例化的概念：

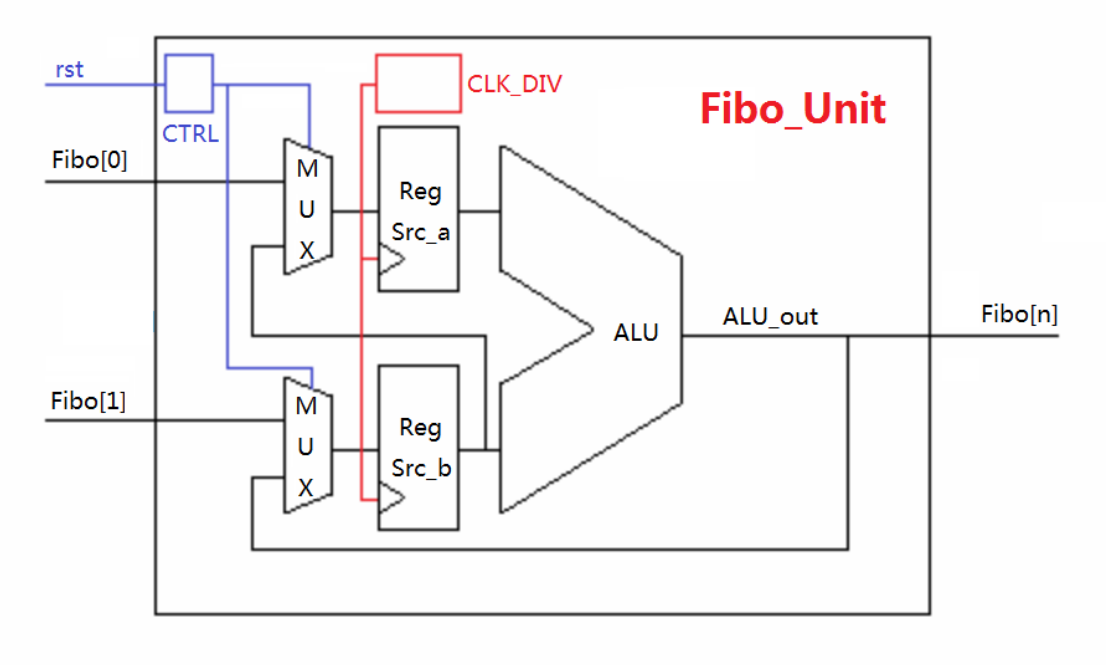
在已经写好 ALU 模块的情况下，在 Fibonacci 模块中实例化时放在 always 块里：

```
always@(posedge clk) begin
    Lab01_ALU ALU_Fibo (…);
    …
end
```

实际上只需要例化一次 ALU 模块即可：

```
Lab01_ALU ALU_Fibo (…);
always@(posedge clk) begin
    …//此处编写对 ALU 的两个操作数的时序逻辑控制代码
end
```

此处给出斐波那契实验的参考模块化结构示意图：



2、 对 verilog 语法的不熟练，如 module 无 I/O、input 信号量定义为 reg 型等

3、 在下载到开发板时，依然采用 initial 语句初始化变量。initial 初始化语句，只能在 testbench 中使用，不可综合。

4、在下载到开发板时，依然采用#开头的时间延时。以#开头的延时不可综合成硬件电路延时，综合工具会忽略所有延时代码，但不会报错。

如：a=#10 b;

这里的#10 是用于仿真时的延时，在综合的时候综合工具会忽略它。也就是说，在综合的时候上式等同于 a=b。

5、在仿真文件中使用 initial 给 f0,f1 赋值，然后在 fib.v 中，再次使用 initial 语句用 f0,f1 给 a,b 赋值。结果造成：f0, f1 是有值的，但是 a,b 不定态。

6、verilog 简介

1） 两种模块接口描述

●

```
module _model (
                    Q  ,
                    CLK ,
                    CEN ,
                    WEN ,
                    A  ,
                    D
);
output [32-1:0]    Q;
input                 CLK;
input                 CEN;
input                 WEN;
input   [32-1:0] A;
input   [32-1:0]  D;
reg  [32-1:0]      Q;
...
endmodule
```

●

```
  module _model (
output reg [32-1:0] Q,
input                 CLK,
input                 CEN,
input                 WEN,
```

```
input   [32-1:0]  A,
input   [32-1:0]  D
                  );
...
endmodule
```

2) 功能描述

Wire—>assign 组合逻辑 阻塞赋值（=）

Reg-->always—>@（*） 组合逻辑 阻塞赋值（=）

Reg-->always-->@(posedge/negedge ） 非阻塞赋值（<=）

7、注意事项

1) Reg 的初始化用：

```
always @(posedge  clk)
   begin
     if ( rst )
      ......
      else
      ......
   end
```

2) 不要交叉使用堵塞、非堵塞赋值。

3) 最好一个 always 只赋值一个 reg。

4) 注意理解设计文件、仿真文件中的语法，不要混用。

5) 理解 IDE 中按钮的功能，它们每个具体的功能。

Synthesis->implementation->programmer