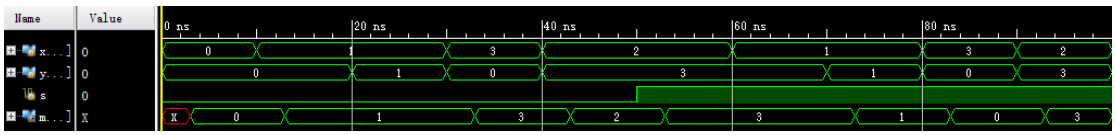
Lab1 问题总结

1. 不理解仿真的作用

仿真用于对设计出来的模块进行测试，通过查看波形图来判断设计有没有错误。拿这次实验的 2-2 的测试文件来举例：

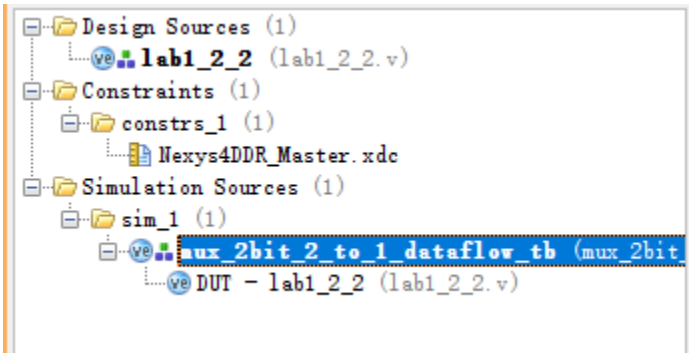
```
module mux_2bit_2_to_1_dataflow_tb(
    );

    reg [1:0] x, y;   //定义了 lab1_2_2 模块的各种输入输出
    reg s;
    wire [1:0] m;

    lab1_2_2   DUT   (.x(x), .y(y), .s(s), .m(m));     //调用 lab1_2_2 模块，验证其正确性


    initial
    begin                    //对 lab1_2_2 模块的输入进行修改，并有一定的时延，以此来验证模
                                块对不对
      x = 0; y = 0; s = 0;
      #10 x = 1;
      #10 y = 1;
      #10 x = 3; y = 0;
      #10 x = 2; y = 3;
      #10 s = 1;
      #10 x = 1;
      #10 y = 1;
      #10 x = 3; y = 0;
      #10 x = 2; y = 3;
    #20;
    end

endmodule
```

执行仿真后就会有如下结果：



这次实现的 lab1_2_2 模块是一个带有 3ns 时延的 2bit 二选一选择器，s 为选择信号，观测输出（m）的值，可以看出 lab1_2_2 模块实现正确。

另外测试文件（这次实验中的 mux_2bit_2_to_1_dataflow_tb.v）需要加入到 simulation sources 文件夹下：

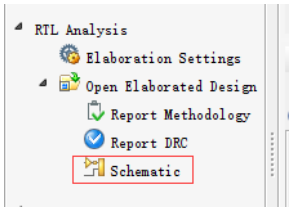


这需要在添加文件时选择 Add or create simulation sources 选项：

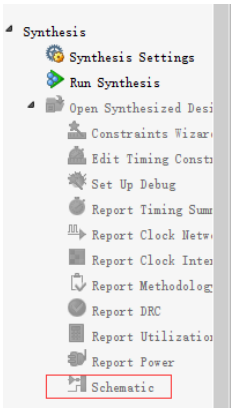
2. 实验过程（个人经验之谈，仅供参考）

实验一般只用执行三步：查看原理图，仿真，下载

（1）一般在实现了自己的模块后，先查看原理图，看看有没有没有连接的线，或者错误的线。比较简单的实验一般不会有太多错，但在实现复杂的项目时，查看原理图可以找到很多错，节省很多时间。一般点击 RTL Analysis 下面的 Schematic：

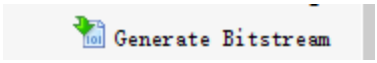


不用 Synthesis 下面的那个：



（2）现在可以添加测试文件进行仿真，来查看自己设计的模块有没有错误，通常是自己设计这个测试文件，并自己给定输入。

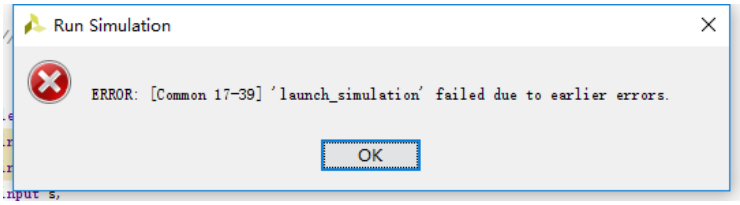
（3）仿真也过了之后就应该没有什么问题了，就可以直接点击 Generate Bitstream 生成二进制文件并下载到板子上：



lab0 里面的很多复杂的环节都不需要，lab0 是了解一下 vivado 怎么用，所以会介绍得很全面，实际上只用上面的那些操作就可以。

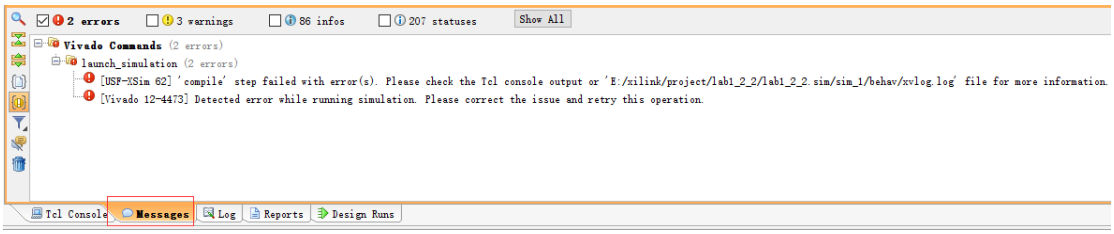
3. Debug

在仿真和 Generate Bitstream 时都会出现错误（生成原理图一般不会报错）：

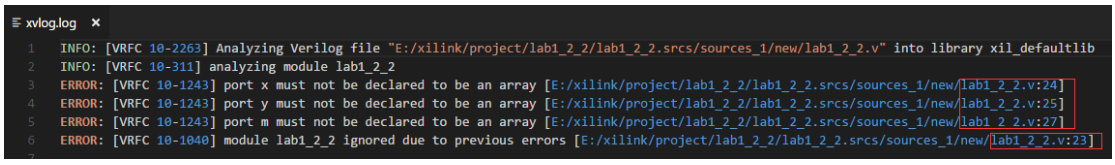


此时在下面的 message 窗口有详细的报错信息：

（1）错误记录在日志文件中

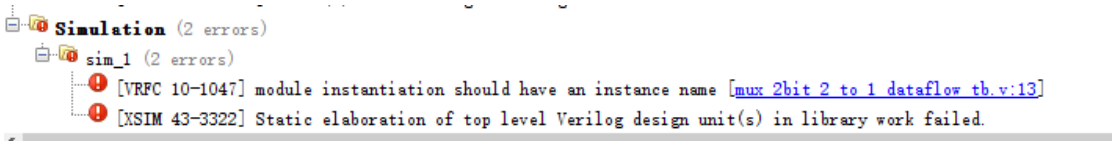


这种报错信息可以在他给出的日志文件下找到错误点：



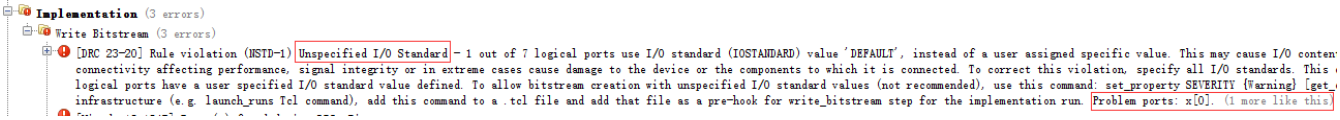
文件中给出了出错的文件和行号，就可以对应的去查找错误。

（2）错误直接写在 message 窗口



也是给出了文件和行号。

（3）管脚错误



管脚错误报错信息很长，在最后会给出出问题的端口，此时可以去查看 xdc 文件，一般和管脚命名及引用有关。

4. 一些其他的小问题

（1）xdc 文件里面不用的管脚要注释掉。

（2）调用模块后要给一个名字：

lab1_2_2　DUT　(.x(x), .y(y), .s(s), .m(m));　　//DUT 就是给这个模块的名字

（3）moudle 和 c 语言的函数很像，但是不能简单的看成是函数，最明显的区别就是：函数调用完了之后就没有了，moudle 只要代码中调用了就一直存在。可以把 moudle 看成一个有输入输出的模块，而在编写 verilog 程序时也就是在搭建一块电路。

（4）数组型变量是不能用在输入输出中的：

```
input  x[1:0],
input  y[1:0],
input s,
output  m[1:0]
```

要用向量型的变量：

```
input [1:0] x,
input [1:0] y,
input s,
output [1:0] m
```