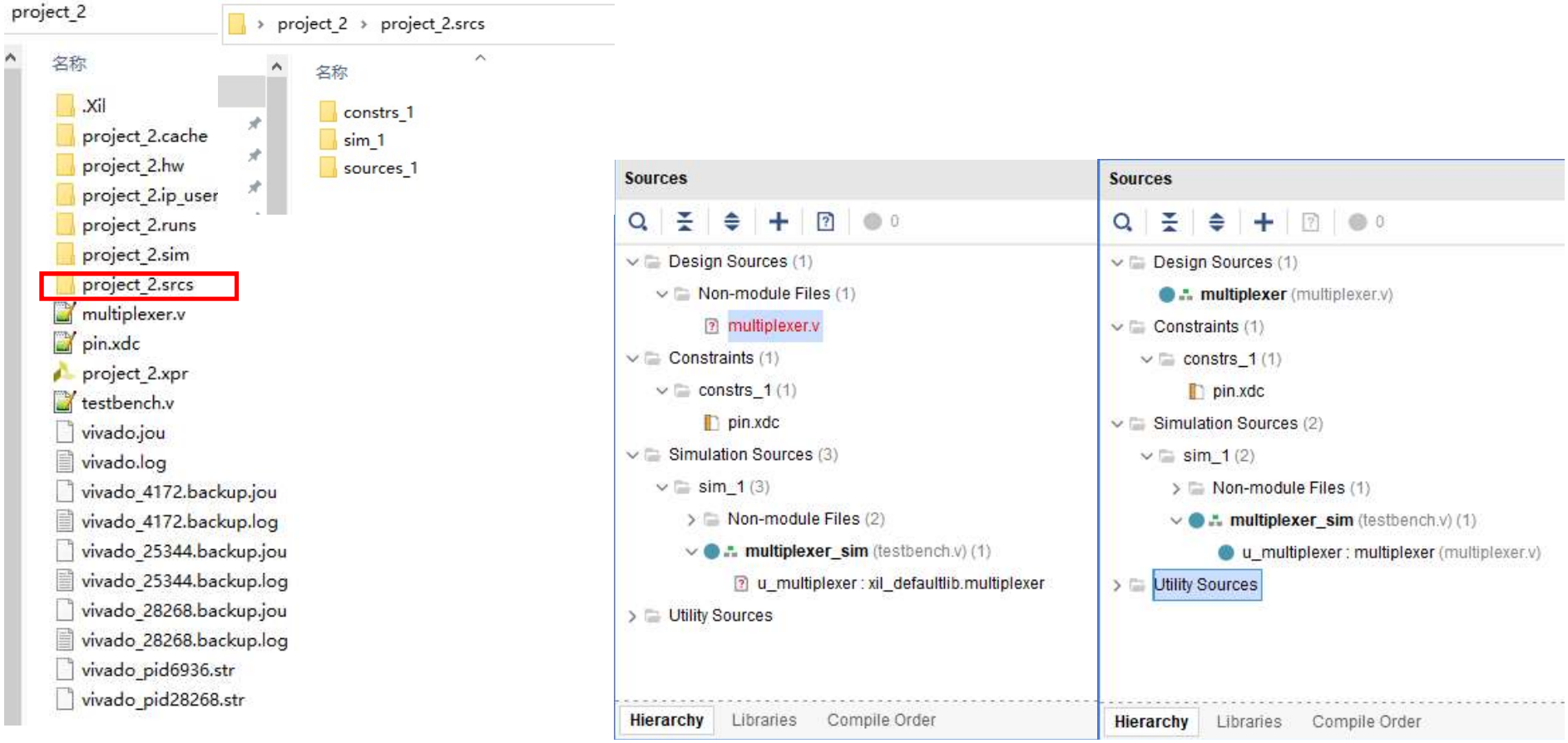
实验一 问题

HITSZ 实验与创新实践教育中心

Education Center of Experiments and Innovations, HITSZ

文件结构


project_2
project_2 › project_2.srcs
名称
.Xil
project_2.cache
project_2.hw
project_2.ip_user
project_2.runs
project_2.sim
project_2.srcs
multiplexer.v
pin.xdc
project_2.xpr
testbench.v
vivado.jou
vivado.log
vivado_4172.backup.jou
vivado_4172.backup.log
vivado_25344.backup.jou
vivado_25344.backup.log
vivado_28268.backup.jou
vivado_28268.backup.log
vivado_pid6936.str
vivado_pid28268.str
名称
constrs_1
sim_1
sources_1
Sources
Design Sources (1)
Non-module Files (1)
multiplexer.v
Constraints (1)
constrs_1 (1)
pin.xdc
Simulation Sources (3)
sim_1 (3)
Non-module Files (2)
multiplexer_sim (testbench.v) (1)
u_multiplexer : xil_defaultlib.multiplexer
Utility Sources
Hierarchy
Libraries
Compile Order
Sources
Design Sources (1)
multiplexer (multiplexer.v)
Constraints (1)
constrs_1 (1)
pin.xdc
Simulation Sources (2)
sim_1 (2)
Non-module Files (1)
multiplexer_sim (testbench.v) (1)
u_multiplexer : multiplexer (multiplexer.v)
Utility Sources
Hierarchy
Libraries
Compile Order

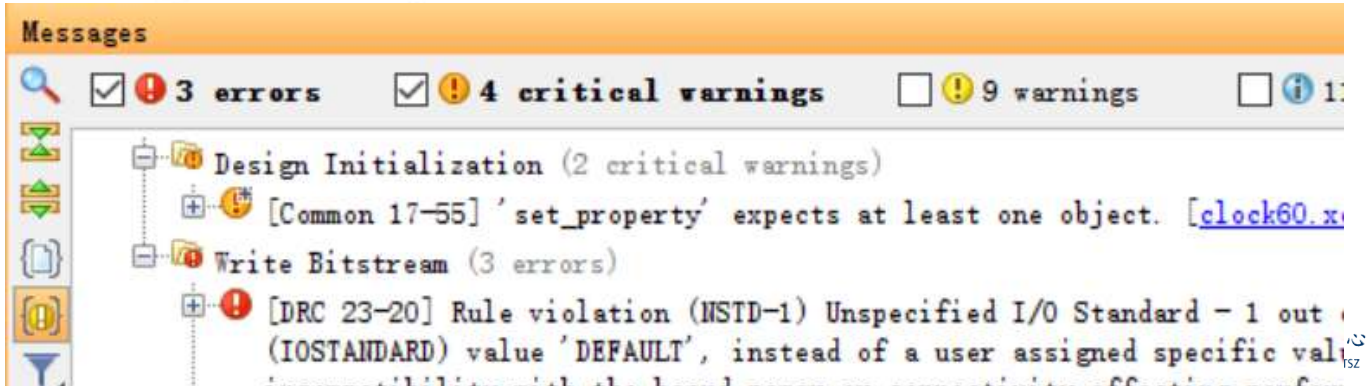
错误提示

- 语法错误提示

```
en0 = (en | 8'hfe);
assign en1 = (en | 8'hfd);
assign en2 = (en |
assign en3 = (en |
assign en4 = (en | 8'hef);
assign en5 = (en | 8'hdf);
assign en6 = (en | 8'hbf);
assign en7 = (en | 8'h7f);
```

Error: Syntax error near "=".
Error: en0 is an unknown type

- Messages窗口



实验二 寄存器文件设计

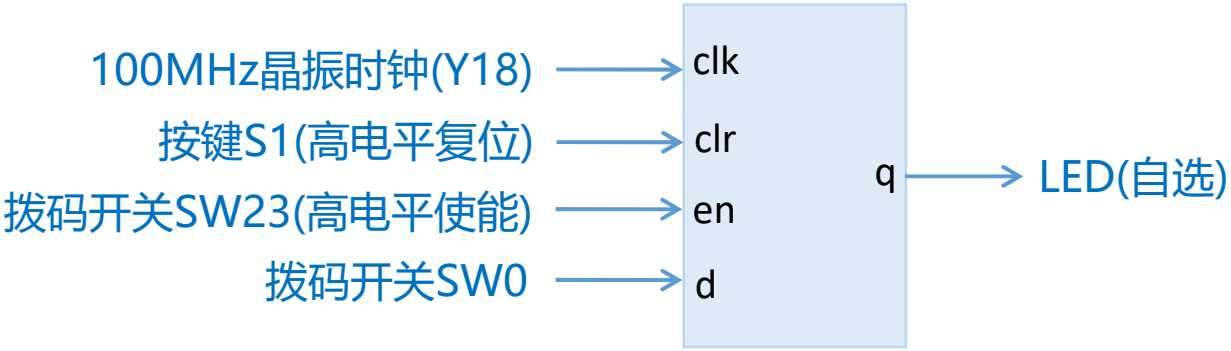
薛睿

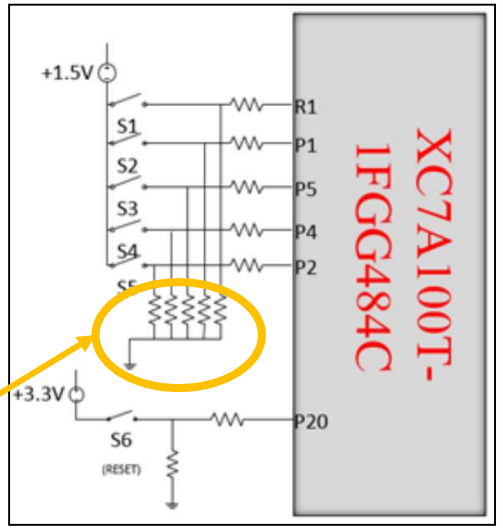
实验目的

- 运用Verilog语言描述时序逻辑电路，理解仿真波形
- 掌握约束文件、仿真文件的编写

实验内容 - 题目1：D触发器

◆ 实现**异步复位**、**同步使能**的D触发器；完成仿真、下板测试





按键接地，默认低电平，按下时高电平

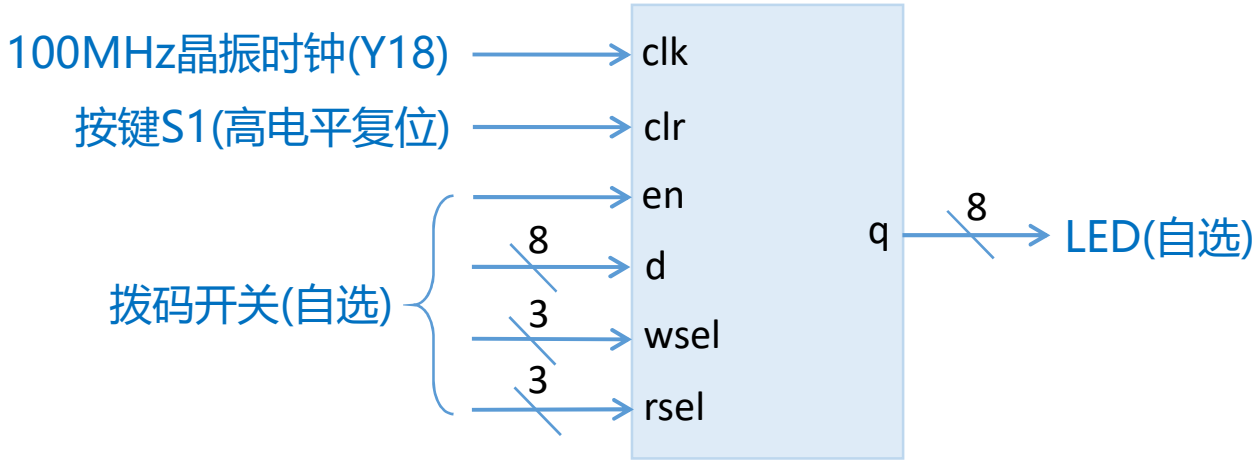
◆ **要求**：

A. 编写仿真文件，需覆盖**写入**(跟随)、**读取**(保持)、**清零**(复位)3种情况

B. 下板测试，独立设计测试步骤，需验证**写入**、**读取**、**清零**3种状态

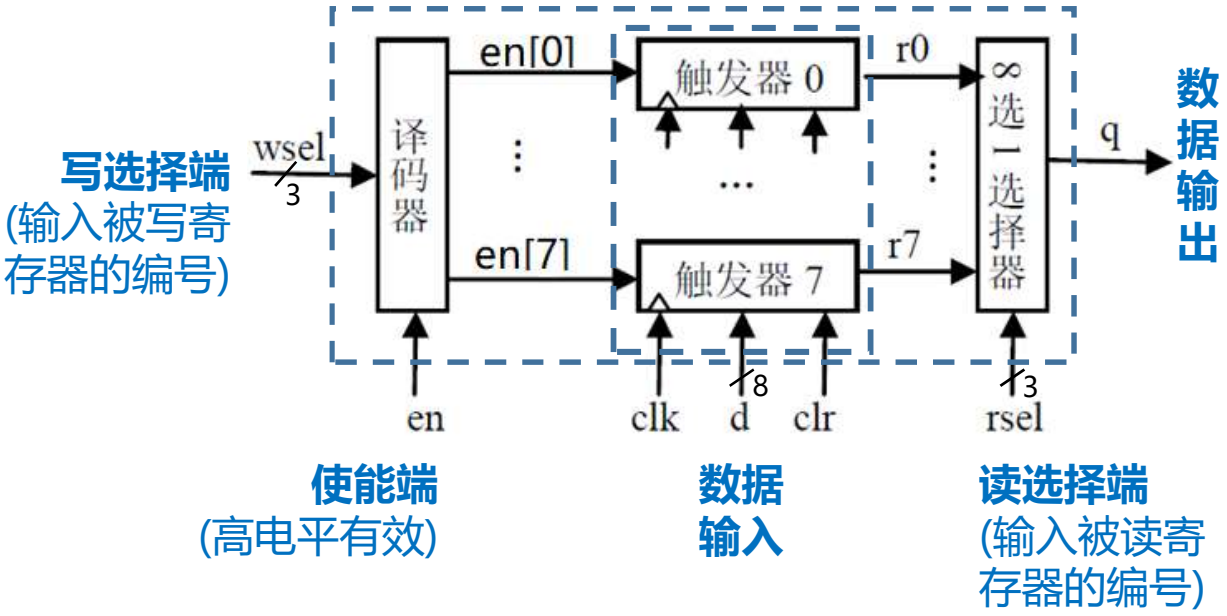
实验内容 - 题目2：8位寄存器堆

- 将D触发器**位宽扩展成8位**，用**8个**8位D触发器加上**译码器**、**多路选择器**，实现8个8位寄存器组成的寄存器堆



实验原理 - 寄存器堆 (Register File)

- 寄存器堆/文件 = **一组**寄存器 + **读写控制**电路，是CPU的核心部件之一
  - 基本功能：读/写某个指定的寄存器



内部信号**en[x]**：

连接触发器x的使能端

内部信号**rx**：

触发器x的输出信号

其他要求

- 1个文件只定义1个module、module名应与文件名一致
- 寄存器堆的实现：
  - 8位D触发器和寄存器堆各一个module，其他module不限
  - 按电路结构图，通过实例化8位D触发器实现
- 时钟clk使用上升沿posedge触发，工程中时钟很少用negedge触发
- 按键开关：一般按一下再马上抬起即可，不需长按！！！
- 所有实验中未注明的细节可自行决定

实验步骤

D触发器 / 8位寄存器堆

① 创建工程，工程名为dff / reg8file

② 编写并添加设计文件dff.v / reg8file.v

③ 根据仿真分析要求，编写并添加仿真文件testbench.v

④ 编写并添加约束文件，并综合实现，生成比特流

⑤ 将生成的比特流下载到开发板验证

Testbench-组合逻辑

```
`timescale 1ns/1ps          //1ns表示延时单位，1ps表示时间精度

module decoder_38_sim();
    reg  [2:0]  data_in;     //3位二进制输入
    reg  [2:0]  en;          //3位使能信号
    wire [7:0]  data_out;    //8位二进制输出

decoder_38 u_decoder_38(     //结合自己的设计模块完成实例化
    .data_in(data_in),
    .en(en),
    .data_out(data_out)
    );

initial begin
      begin en = 3'b100;data_in = 3'b000;end  //构造输入激励信号
    #5 begin en = 3'b100;data_in = 3'b001;end
    #5 begin en = 3'b100;data_in = 3'b010;end
    #5 begin en = 3'b100;data_in = 3'b011;end
    #5 begin en = 3'b100;data_in = 3'b100;end
    #5 begin en = 3'b100;data_in = 3'b101;end
    #5 begin en = 3'b100;data_in = 3'b110;end
    #5 begin en = 3'b100;data_in = 3'b111;end
    #5 begin en = 3'b000;data_in = 3'b000;end  //使能端无效
    #5 begin en = 3'b101;data_in = 3'b111;end  //使能端无效
    #5 $stop;  //立即结束仿真
end

endmodule
```

- 仿真模块没有输入、输出端口
- 激励信号数据类型要求为reg，以便保持激励值不变，直至执行到下一跳激励语句为止
- 输出信号数据类型要求为wire，以便能随时跟踪激励信号的变化
- 实例化设计模块
- initial块只能执行一次，不带触发条件。

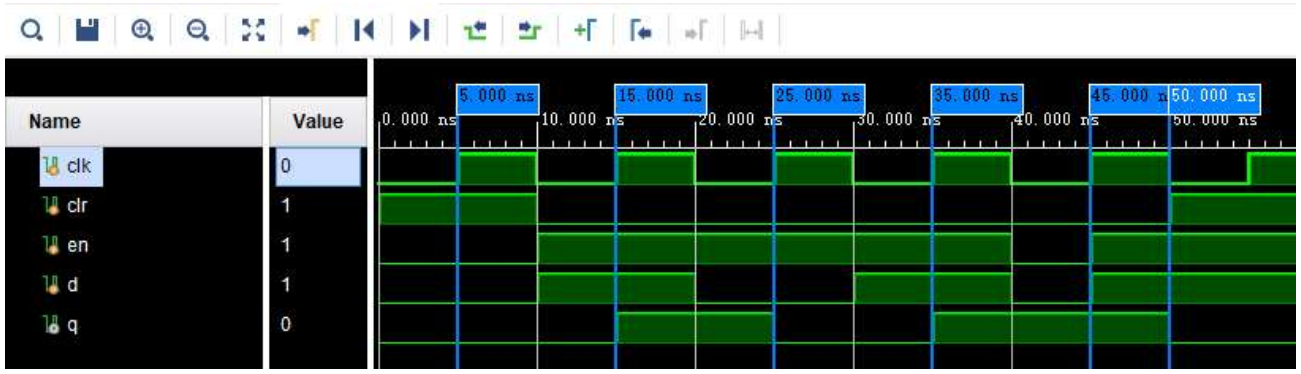
时钟信号的仿真

时间单位

```
`timescale 1ns/1ps

module testbench (
);

reg clk;

initial begin
    clk = 0;
end

always #5 clk = ~clk;

endmodule
```

5个时间单位

- 定义reg型变量的时钟信号clk
- 在initial块初始化clk为0
  - 或定义时初始化（`reg clk = 0; // 仅限仿真`）
- 在always块中，周期性令clk信号翻转



每5ns翻转一次，即周期为10ns，故clk频率为100MHz

课后作业

- always 块被触发时信号取的什么时刻的值
  - 编写testbench对所给代码进行仿真
  - 结合波形分析做出回答

```
always @(posedge clk)
begin
    if(rst)
        a <= 8'd0;
    else
        a <= 8'd2;
end
```

```
always@(posedge clk)
begin
    if(rst)
        b <= 8'd0;
    else if(a == 8'd2)
        b <= 8'd3;
end
```

```
always@(posedge clk)
begin
    if(rst)
        c <= 8'd0;
    else if(b == 8'd3)
        c <= 8'd4;
end
```

验收&提交要求

- 课上检查
  - ① D触发器仿真、上板验证，检查内容：

    写入/跟随、读取/保持、清零/复位
  - ② 寄存器堆上板验证，检查内容：

    先向2个不同编号的寄存器写入2个随机数据，再按写入顺序读取
- 课后提交
  - 寄存器堆仿真波形分析、代码、RTL 分析和综合后原理图截图
  - 课后作业的仿真分析说明

验收&提交要求

- 提交时间：**详见作业提交系统**
- 提交格式：**学号_姓名.zip**
- 注意：如有雷同，**雷同者均0分！**

开始实验

HITSZ 实验与创新实践教育中心

Education Center of Experiments and Innovations, HITSZ

时序逻辑电路的Verilog描述

时钟沿触发的always块描述

```
always @（<敏感信号列表>）
   begin
        //过程赋值
        //if-else、case选择语句
        //for、while等循环块
    end
```

边沿触发：时钟处在上升沿或下降沿时，语句被执行。

always @(posedge clk) ——时钟上升沿触发

always @(negedge clk) ——时钟下降沿触发

always @(posedge clk or negedge rst_n) 带异步复位的时钟上升沿触发

用Verilog实现带异步清零端的D触发器

```
module VrDffC(input CLK, input CLR, input D,
                    output reg Q);
  always @ (posedge CLK or posedge CLR)
    if (CLR==1) Q <= 0;
    else Q <= D;
endmodule
```

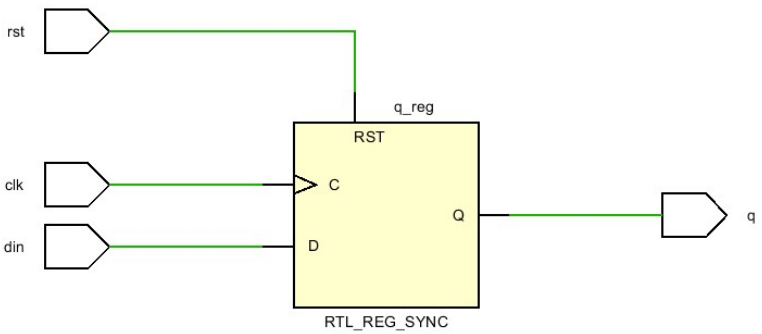
带时钟使能端和同步置位D触发器

```
module VrDffSE(input CLK, input S, input CE,
               input D, output reg Q);
  always @ (posedge CLK)
    if (S==1) Q <= 1;
    else if (CE==1) Q <= D;
    else Q <= 0;
endmodule
```

时序逻辑的Verilog描述

- 时序逻辑电路的变化通过时钟沿触发，需要使用时钟沿触发的always块描述。
- 用always块描述时序逻辑电路时，用非阻塞赋值。

```
//带同步复位的D触发器
module dff_r(
        input    wire    clk,
        input    wire    rst,
        input    wire    din,
        output   reg     q
);
always@(posedge clk)begin
        if(rst)     q <= 1'b0;
        else        q <= din;
end
endmodule
```



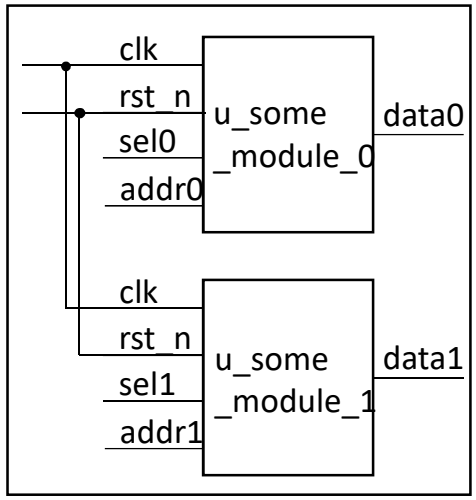
模块实例化


HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

从“模块”开始搭电路-模块实例化

- 一个模块在另外一个模块中实例化，等效于实际电路中加入了被实例化的电路。
- 模块实例用u_x_x表示（多次实例化用序号0、1、2等表示）；

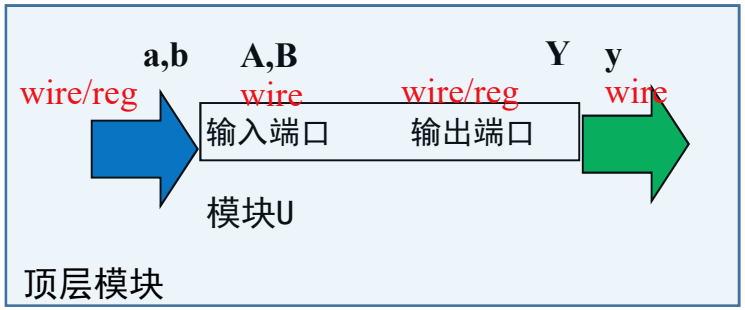
```
module_name instance_name(
.port_name1(data_name1),
.port_name2(data_name2),
……);
```

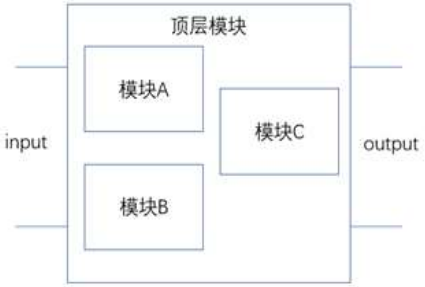



```
wire        sel0 ;
wire        sel1 ;
wire [3:0] addr ;
wire        data0;
wire        data1;

some_module u_some_module_0 (
    .clk    (clk  ),
    .rst_n  (rst_n),
    .sel    (sel0 ),
    .addr   (addr ),
    .data   (data0)
);

some_module u_some_module_1 (
    .clk    (clk  ),
    .rst_n  (rst_n),
    .sel    (sel1 ),
    .addr   (addr ),
    .data   (data1)
);
```

如何选择正确的数据类型？

- 端口：（input、output）
- 可以由reg或wire连接驱动，但它本身只能驱动wire连接。



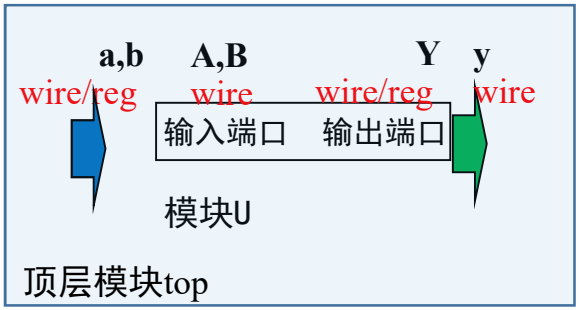


- 变量：
- assign赋值语句左侧数据类型，必须是wire类型；
- 过程块（always块 或 initial块）中被赋值的左侧数据类型，必须把它声明为reg类型变量。

如何选择正确的数据类型？

模块U：

```
module U(
output wire Y,
input wire A, B);

//功能描述

endmodule
```



模块实例化不是参数传递

模块端口与外部信号按照其名字进行连接

顶层模块：

```
module top(
端口定义
) ;
wire y;
reg a, b;

U u1(
.Y(y),
.A(a),
.B(b)
);

//其他逻辑
endmodule
```

- 最后一排没有板子，不要做最后一排