南京理工大学计算机科学与工程学院

计算机逻辑基础 实验报告

班　　级 9181069502

学生姓名 黄海浪

学　　号 9181040G0818

教　　师 余立功

教　　师 潘志兰

目录

实验一 8-3 编码器实现

一、实验目的

1. 熟悉常用编码器，译码器的功能逻辑。
2. 学习组合逻辑电路译码器或编码器的设计方法及应用。
3. 熟悉 Verilog 代码的编写方法。
4. 掌握复杂编码译码器的设计方法。

二、实验内容

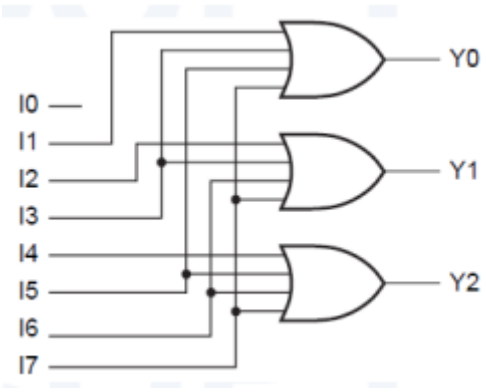
主要内容：用 Verilog 来实现 8-3 编码器并实现 8-3 编码器的仿真。

实验原理：

编码器是指能将每一组输入信息或数目变换为相应二进码输出，即能完成特定编码功能的逻辑电路。常用的编码器有二进制编码器、二-十进制编码器、优先编码器等。

用 n 位二进制代码对 2^n 个信号进行编码的电路，称为二进制编码器。常用的二进制编码器有 4 线-2 线、 8 线-3 线和 16 线-4 线等。由于编码器各个输出信号逻辑表达式的基本形式是有关输入信号的或运算，所以其逻辑电路是由或门组成的阵列，这也是编码器基本电路结构的一个显著特点。

8-3 编码器原理图：



8-3 编码器真值表：

输入din[7:0]								输出dout[2:0]		
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1
其他								0	0	0

逻辑表达：

dout = max（二进制）;其中 din[max]==1，且 din[max+1]=0

三、实验步骤

（1） 根据 Vivado 的设计流程，先建立本次实验项目的工程文件；
（2） 建立本次实验内容的 Verilog 功能模块，修改语法错误；
（3） 建立仿真文件，进行电路的功能仿真；
（4） 编译综合，直到没有错误，查看综合结果。

四、实验程序

源文件：

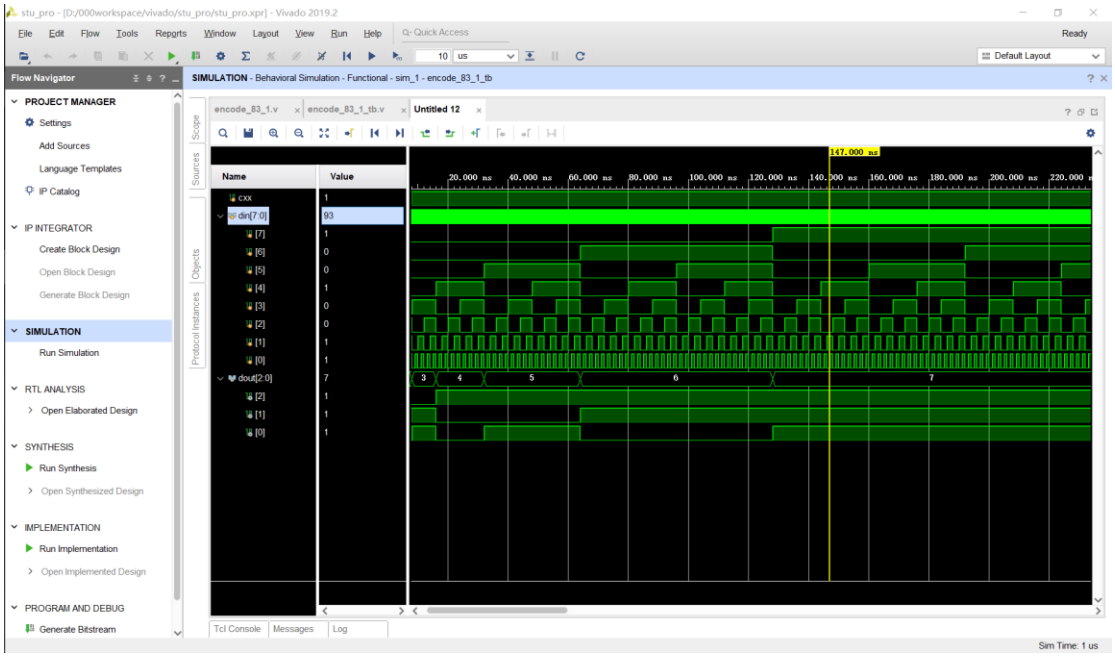
```
//encode_83_1
module encode_83_1(din,dout);     //参数
    input wire[7:0] din;         //输入
    output reg[2:0] dout = 3'b000;        //输出
    integer i=0;                //遍历位数使用
    always @(din) begin            //输入改变时
        for (i=0;i<8;i=i+1) begin
            if (din[i]==1) dout<=i;       //通过 i 位获取值 (最终获取到最大
的)
        end
    end
endmodule
```

仿真文件：

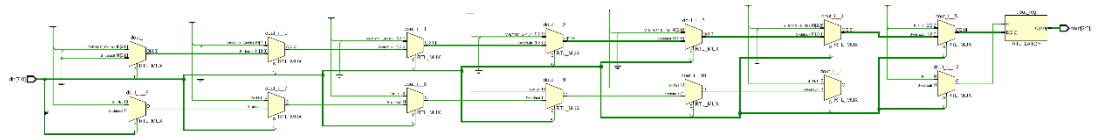
```
//encode_83_1_tb
module encode_83_1_tb;
    reg cxx;         //cxx 为判断 din 是否达到最大，达到则回退
    reg [7:0] din;  //输入
    wire [2:0] dout;//输出
    encode_83_1 E83(.din(din),.dout(dout));//关联
    initial begin   //初始化
        cxx = 1'b1; //判断是否 11111111
        din = 8'b00000000;
        forever begin   //重复
            #1 if(cxx==1'b1) din = din + 1; //间隔 1
            else din = din - 1;
            if(din==8'b11111111) cxx = 1'b0;
            if(din==8'b00000000) cxx = 1'b1;
        end
    end
endmodule
```

五、实验结果

仿真图：

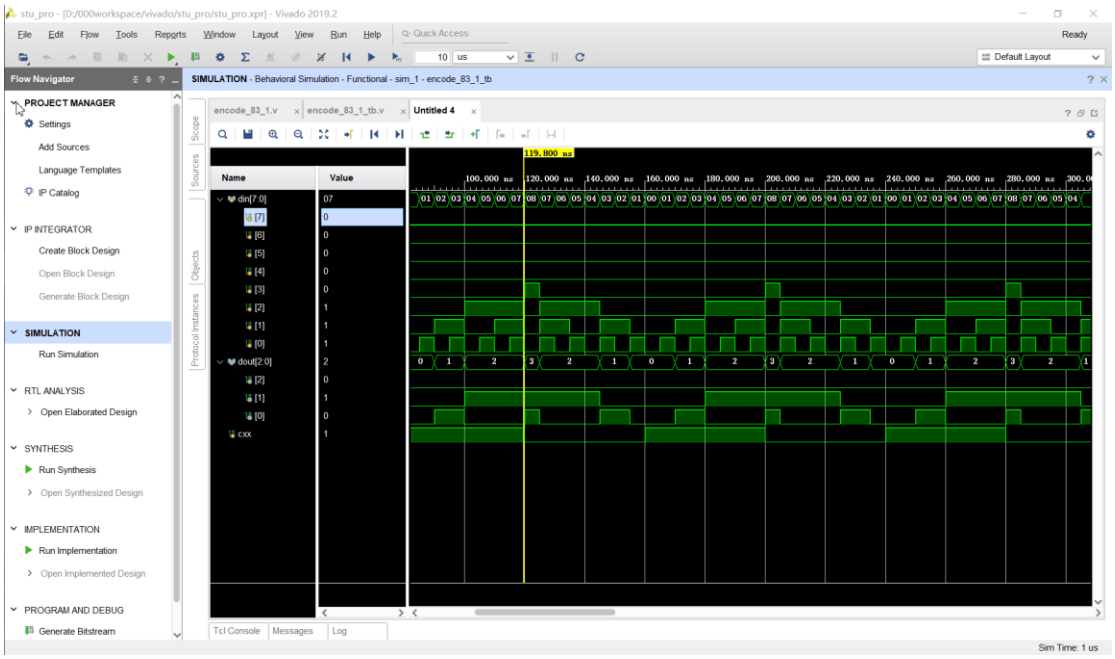


生成原理图：

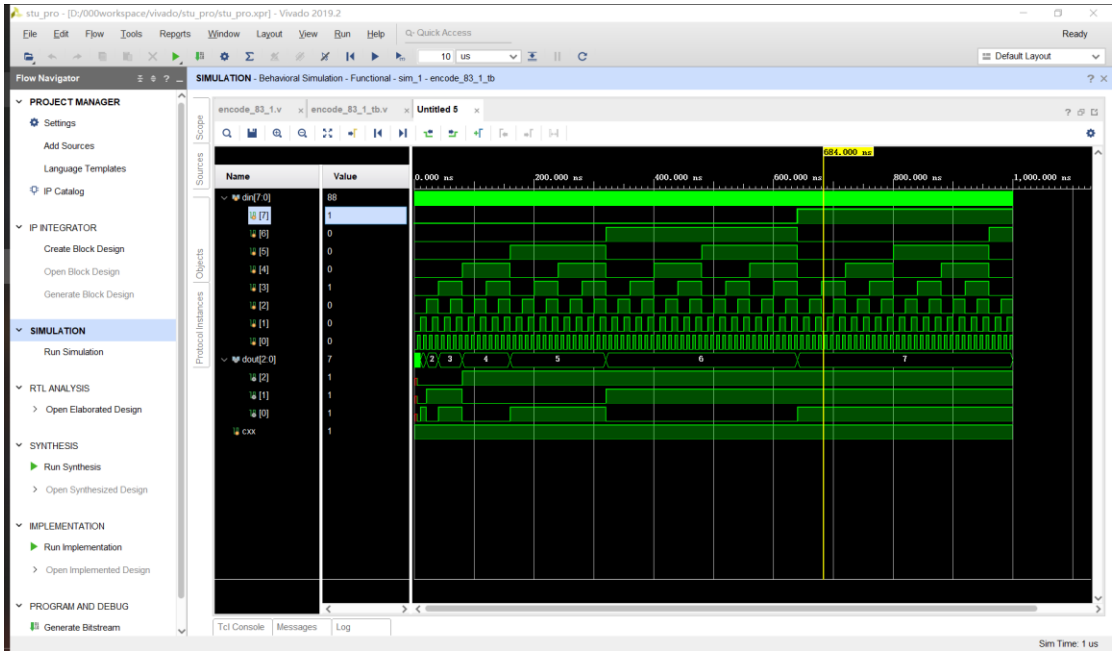


六、实验调试问题与解决办法

问题 1：数据格式错误
问题解决方案：不断调试，更改数据格式以及数据赋值等。



问题 2：时间延迟控制未控制好
问题解决方案：多次仿真，调试时间间隔，直到得到一个不错的结果。



问题 3：前面 initial 的时候未赋初值，所以有红色的部分
问题解决方案：对 dout 赋初值为 0

实验二 8 选 1 数据选择器实现

一、实验目的

1. 熟悉常用电路选择器的功能逻辑。
2. 学习组合逻辑电路数据选择器的设计方法及应用。
3. 熟悉 Verilog 代码的编写方法。
4. 掌握复杂电路数据选择器的设计方法。

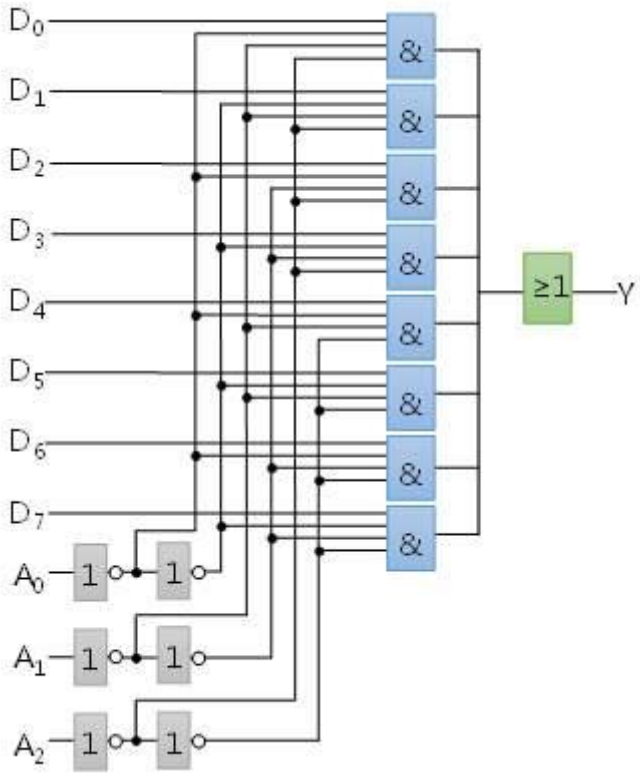
二、实验内容

主要内容：用 Verilog 来实现 8 选 1 数据选择器并实现 8 选 1 数据选择器的仿真。

实验原理：

在数字系统中，经常需要把多个不同通道的信号发送到公共的信号通道上，通过多路选择器可以完成这一功能。在数字系统设计中，常用 CASE 语句和 IF 语句描述多路选择器。多路选择器是由几路数据输入、一位或多位的选择控制，和一路数据输出所组成的，如 2 选 1、4 选 1、8 选 1 等多路选择器。

8 选 1 数据选择器原理图：



8 选 1 数据选择器真值表：

	sin[2:0]			dout
din[7:0]	0	0	0	din[0]
	0	0	1	din[1]
	0	1	0	din[2]
	0	1	1	din[3]
	1	0	0	din[4]
	1	0	1	din[5]
	1	1	0	din[6]
	1	1	1	din[7]

逻辑表达：
dout = din[sin]; 其中 sin[2:0]转为十进制。

三、实验步骤

（1） 根据 Vivado 的设计流程，先建立本次实验项目的工程文件；
（2） 建立本次实验内容的 Verilog 功能模块，修改语法错误；
（3） 建立仿真文件，进行电路的功能仿真；
（4） 编译综合，直到没有错误，查看综合结果。

四、实验程序

源文件：

```
//select_81_2
module select_81_2(sin,din,dout);
    input wire[2:0] sin;            //输入
    input wire[7:0] din;            //输入
    output reg dout = 1'bx;           //输出
    always @(sin or din) begin        //输入改变时
    dout = din[sin];
  end

endmodule
```

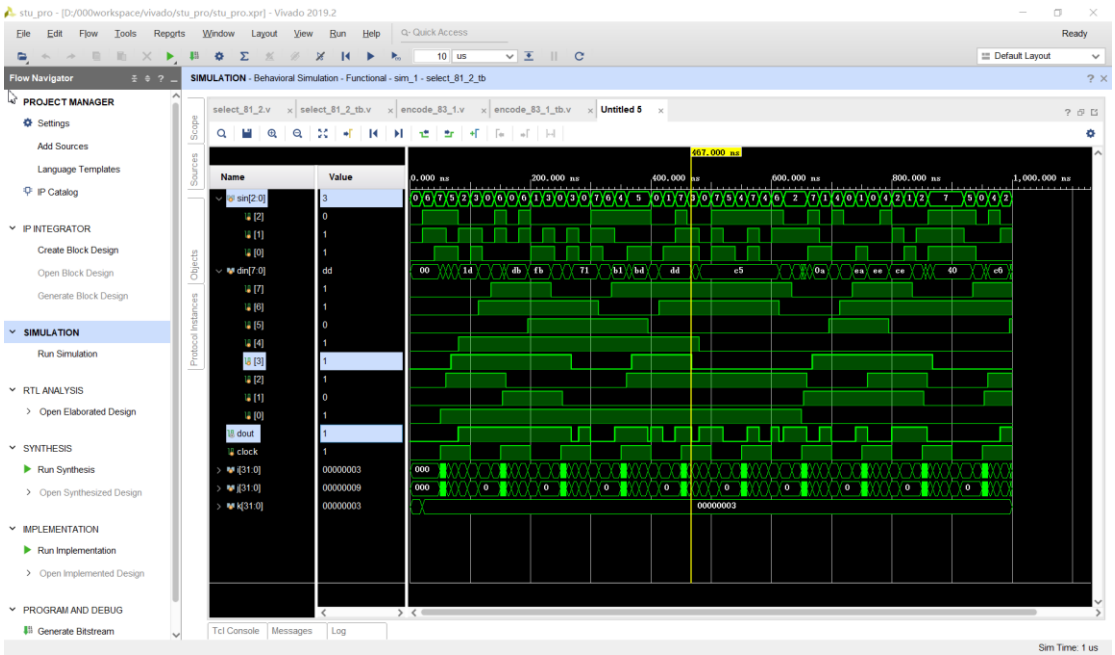
仿真文件：

```
//select_81_2_tb
module select_81_2_tb();
    reg[2:0] sin;             //输入
    reg[7:0] din;             //输入
    wire dout;                //输出
    reg clock;                //这次使用 clock 产生信号
    select_81_2 S81(.sin(sin),.din(din),.dout(dout));
```

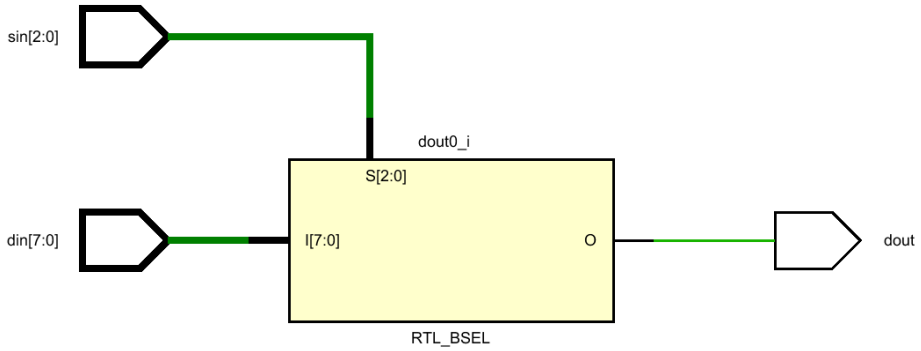
```
    initial begin
        din = 8'b00000000;
        sin = 8'b000;
        clock = 0;
    end
    always #50 clock = ~clock;       //50 单位变化 clock
    integer i = 0,j = 0;            //i j 数组使用 时间延迟
    always @(posedge clock) begin    //输入更改
        for(i = 0;i<8;i=i+1) begin
            j = i*3;
            #j din[i] = {$random}%2;     //随机数
        end
    end
    integer k = 0;
    always begin
        #20 for(k=0;k<3;k=k+1)begin      //随机赋值
           sin[k] = {$random}%2;
        end
    end
endmodule
```

五、实验结果

仿真图：

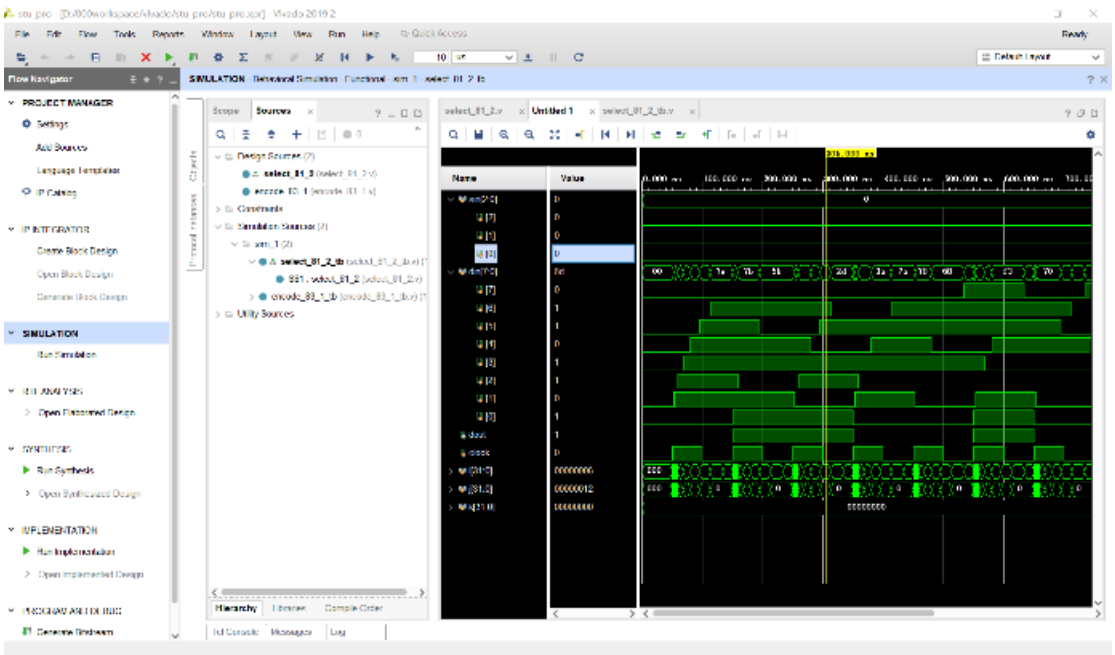


生成原理图：

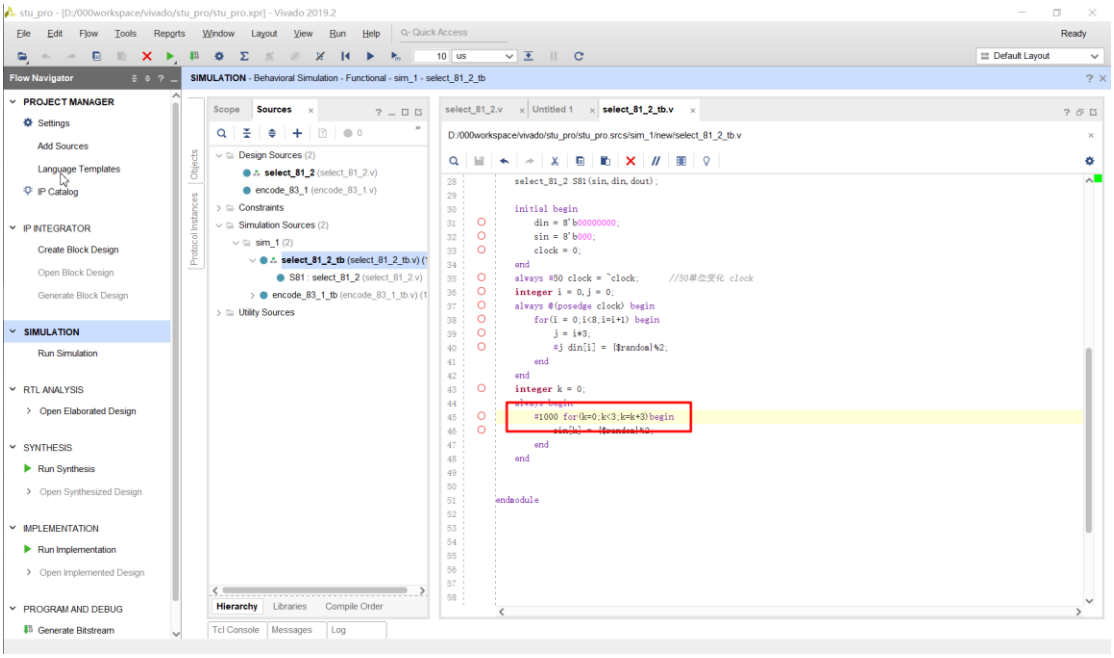


六、实验调试问题与解决办法

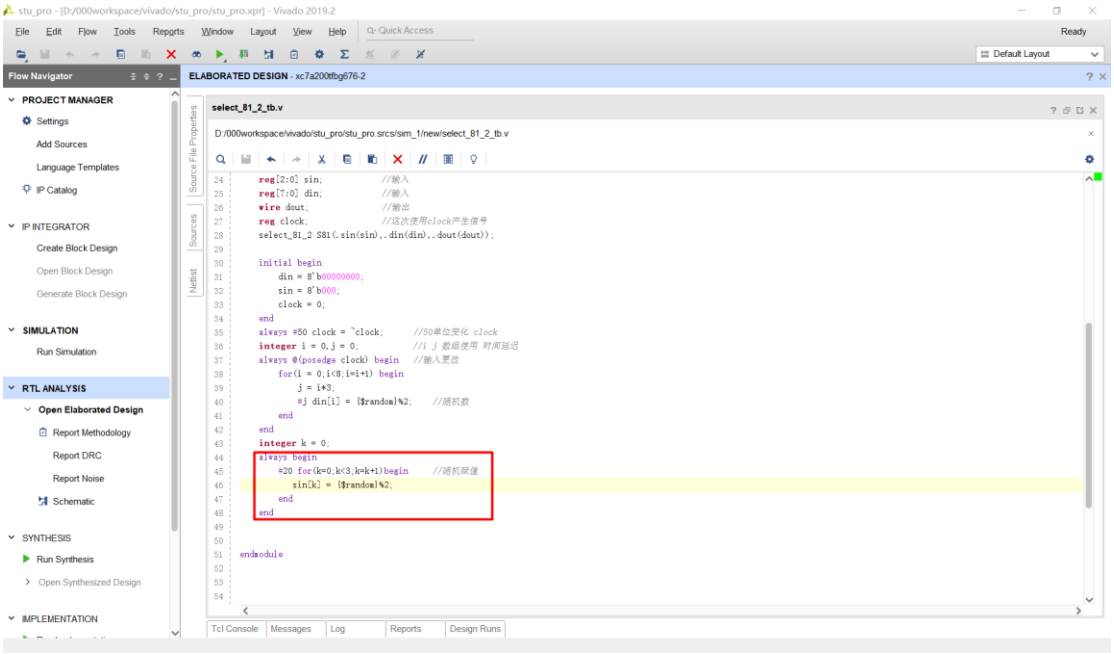
问题 1：sin 无变化，如下图



问题查找：


stu_pro - [D:/000workspace/vivado/stu_pro/stu_pro.xpr] - Vivado 2019.2
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Ready
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
SIMULATION - Behavioral Simulation - Functional - sim_1 - select_81_2_tb
Scope
Sources
Design Sources (2)
select_81_2 (select_81_2.v)
encode_83_1 (encode_83_1.v)
Constraints
Simulation Sources (2)
sim_1 (2)
select_81_2_tb (select_81_2_tb.v)
S81 : select_81_2 (select_81_2.v)
encode_83_1_tb (encode_83_1_tb.v)
Utility Sources
Hierarchy
Libraries
Compile Order
select_81_2.v
Untitled 1
select_81_2_tb.v
D:/000workspace/vivado/stu_pro/stu_pro.srcs/sim_1/new/select_81_2_tb.v
select_81_2 S81(sin, din, dout);
initial begin
din = 8'b00000000;
sin = 8'b000;
clock = 0;
end
always #50 clock = ~clock; //50单位变化 clock
integer i = 0, j = 0;
always @(posedge clock) begin
for(i = 0;i<8;i=i+1) begin
j = i*3;
#j din[i] = {$random}%2;
end
end
integer k = 0;
always begin
#1000 for(k=0;k<3;k=k+3)begin
sin[k] = {$random}%2;
end
end
endmodule
Tcl Console
Messages
Log

错误修改：


stu_pro - [D:/000workspace/vivado/stu_pro/stu_pro.xpr] - Vivado 2019.2
File Edit Flow Tools Reports Window Layout View Help
Quick Access
Ready
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
Report Methodology
Report DRC
Report Noise
Schematic
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
ELABORATED DESIGN - xc7a200tfbg676-2
select_81_2_tb.v
D:/000workspace/vivado/stu_pro/stu_pro.srcs/sim_1/new/select_81_2_tb.v
reg[2:0] sin; //输入
reg[7:0] din; //输入
wire dout; //输出
reg clock; //这次使用clock产生信号
select_81_2 S81(.sin(sin),.din(din),.dout(dout));
initial begin
din = 8'b00000000;
sin = 8'b000;
clock = 0;
end
always #50 clock = ~clock; //50单位变化 clock
integer i = 0,j = 0; //i j 数组使用 时间延迟
always @(posedge clock) begin //输入更改
for(i = 0;i<8;i=i+1) begin
j = i*3;
#j din[i] = {$random}%2; //随机数
end
end
integer k = 0;
always begin
#20 for(k=0;k<3;k=k+1)begin //随机赋值
sin[k] = {$random}%2;
end
end
endmodule
Tcl Console
Messages
Log
Reports
Design Runs

实验三 环形右移寄存器实现

一、实验目的

1. 熟悉常用时序逻辑电路计数器的功能逻辑。
2. 学习时序逻辑电路计数器的设计方法及应用。
3. 熟悉 Verilog 代码的编写方法。
4. 掌握复杂时序逻辑电路计数器的设计方法。

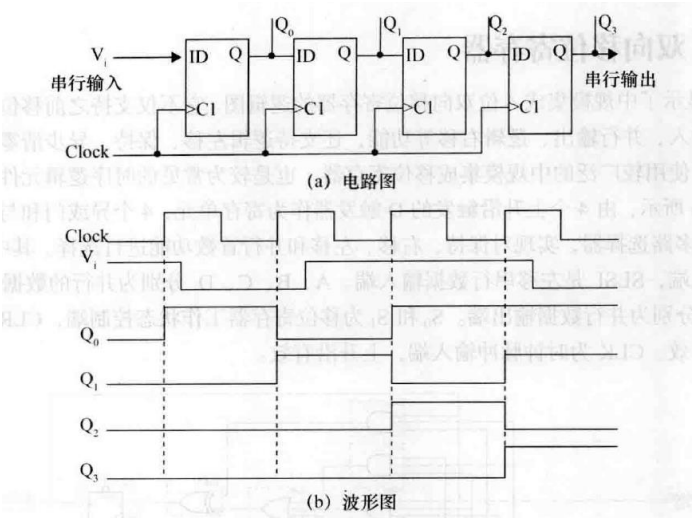
二、实验内容

主要内容：用 Verilog 来实现环形右移位寄存器并实现环形右移位寄存器的仿真。

实验原理：

在数字系统设计中，计数器和寄存器是最常用的两类功能器件，也是最常见的时序逻辑元件。计数器是一种能统计输入脉冲个数的时序电路，它可以记录特定事件的发生次数，产生控制系统中不同任务的时间间隔，记录特定事件之间的时间间隔等，它可以用于定时器、分频器、程序控制器、信号发生器等多种数字设备中。计数器按脉冲的作用方式分类，可分为同步计数器和异步计数器。在同步计数器中，各个触发器的时钟输入端和同一个时钟脉冲源相连，因而所有触发器状态（即计数器状态）的改变都与时钟脉冲同步。而在异步计数器中，有的触发器直接受输入计数脉冲控制，有的是利用其他触发器输出作为时钟输入信号，因此所有触发器状态的改变有先有后，是异步的。

右移位寄存器原理图：



环形右移位寄存器真值表：

输入	clk	dout[9:0]
din[9:0]	↑	din右移

逻辑表达：
dout = {din[0], din[9:1]}；当 clk 上升沿到来时。

三、实验步骤

（1） 根据 Vivado 的设计流程，先建立本次实验项目的工程文件；
（2） 建立本次实验内容的 Verilog 功能模块，修改语法错误；
（3） 建立仿真文件，进行电路的功能仿真；
（4） 编译综合，直到没有错误，查看综合结果。

四、实验程序

源文件：

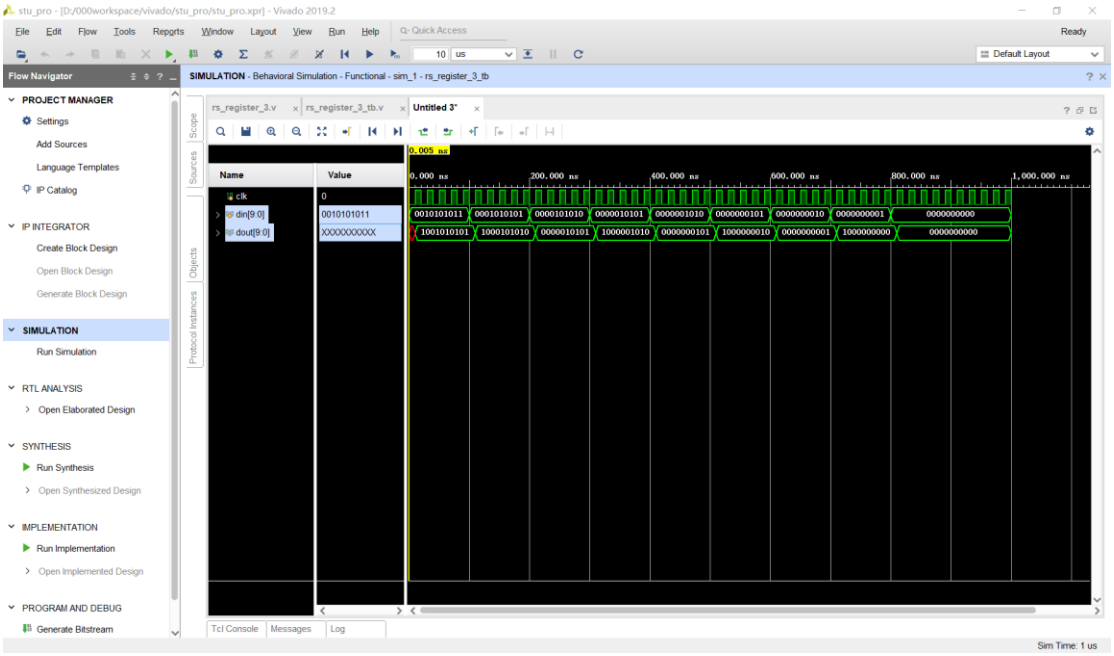
```
//rs_register_3
module rs_register_3(clk, din, dout);   //输入输出
    input wire clk;
    input wire [9:0] din;
    output reg [9:0] dout;

    always@(posedge clk)  dout <= {din[0], din[9:1]};   //计算移位后，环形右移

endmodule
```

仿真文件：

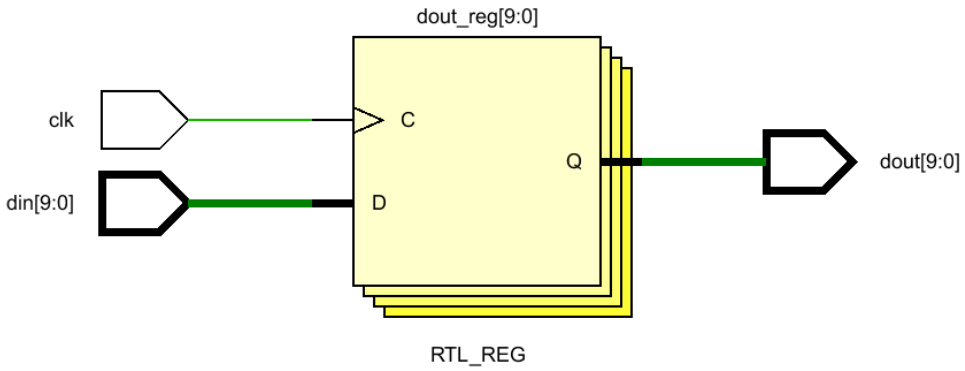
```
//rs_register_3_tb
module rs_register_3_tb();
    reg clk;
    reg [9:0] din;
    wire [9:0] dout;

    always #10 clk = ~clk;  //clock 10 ns 迭代
    rs_register_3 RSR(.clk(clk),.din(din), .dout(dout));    //关联
    initial begin
        clk = 1'b0;         //clock 初始化
        din = 10'b0010101011;   //初始化
        forever #100 din <= din>>1; //迭代 将 din 不断右移
    end
endmodule
```

五、实验结果

仿真图：



生成原理图：



六、实验调试问题与解决办法

当前实验因为移位运算较为简单，并且有前面两个实验的基础，并未出现任何错误。

实验四 0101 mealy 序列检测状态机实现

一、实验目的

1. 熟悉常用时序逻辑电路中 Mealy 和 Moore 状态机的功能逻辑。
2. 学习时序逻辑电路中 Mealy 和 Moore 状态机的设计方法及应用。
3. 熟悉 Verilog 代码的编写方法。
4. 掌握复杂时序逻辑电路中 Mealy 和 Moore 状态机的设计方法。

二、实验内容

主要内容：用 Verilog 来实现 Mealy 型 0101 序列检测状态机并实现仿真。

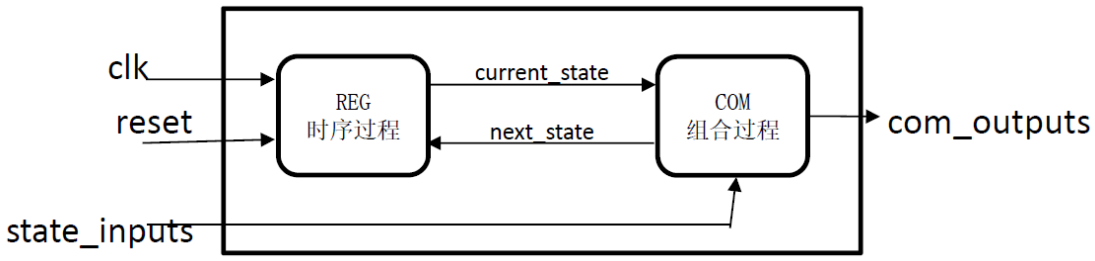
实验原理：

有限自动状态机 FSM（Finate State Machine)是复杂数字系统设计中非常重要的一部分，是实现高效率高可靠性逻辑控制的重要途径。大部分数字系统都是由控制单元和数据单元组成的。数据单元负责数据的处理和传输，而控制单元主要是控制数据单元的操作顺序。在数字系统中，控制单元往往通过使用有限状态机实现，有限状态机接受外部信号以及数据单元产生的状态信号，从而产生控制信号序列。

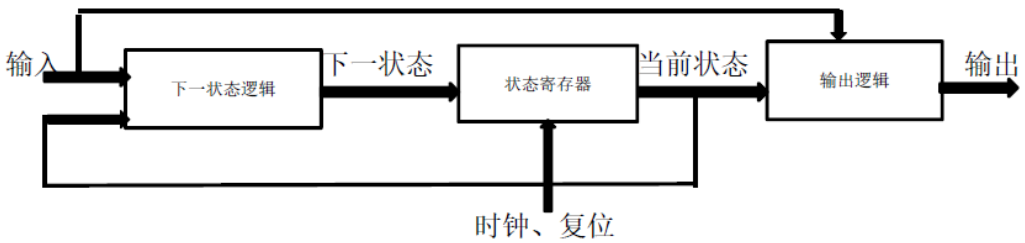
状态机分类：

① 从状态机的信号输出方式上分，有 Mealy 型和 Moore 型两种状态机。
② 从状态机的描述结构上分，有单过程状态机多过程状态机。
③ 从状态表达方式上分，有符号化状态机和确定状态编码的状态机。
④ 从状态机编码方式上分，有顺序编码状态机、一位热码编码状态机或其他编码方式状态机。

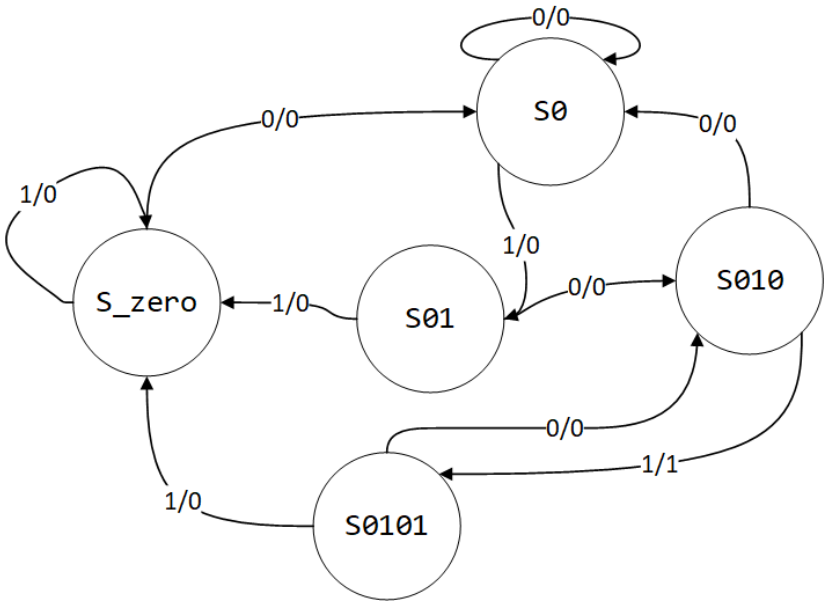
有限状态机的一般结构图：



Mealy 型状态机如下图：



Mealy 型 0101 序列状态机状态转移图：



三、实验步骤

（1） 根据 Vivado 的设计流程，先建立本次实验项目的工程文件；
（2） 建立本次实验内容的 Verilog 功能模块，修改语法错误；
（3） 建立仿真文件，进行电路的功能仿真；
（4） 编译综合，直到没有错误，查看综合结果。

四、实验程序

原文件：

```
// mealy_4
// 0101 序列检测
module mealy_4(in, clk, rst,mealy_out,state,state_next);
input wire in, clk, rst;    //输入 时钟 置零
output reg  mealy_out;      //输出
output reg [3:0] state,state_next;    //状态 下一个状态
//定义状态
localparam  S_zero = 4'b1111,
            S0 = 4'b1110,
            S01 = 4'b1101,
            S010 = 4'b1010,
            S0101 = 4'b0101;

always @(posedge clk or posedge rst) begin
    if (rst) state <= S_zero;   //是否置零
    else state <= state_next;   //否则转化为下一个状态
```

```
end

always @(*) begin   //任意输入变化
    if (rst) state_next = S_zero;   //是否置零
    else begin
        case(state) //检查状态 以便获取下一个状态
            S_zero: state_next = (in == 0) ? S0:S_zero;
            S0: state_next = (in == 1) ? S01 :  S0;
            S01: state_next =  (in == 0) ? S010 : S_zero;
            S010: state_next = (in == 1) ? S0101 : S0;
            S0101: state_next = (in ==0) ? S010 : S_zero;
            default: state_next = S_zero;   //默认为初始值
        endcase
    end
end


always @(posedge clk or posedge rst) begin
    if (rst) mealy_out <= 1'b0;     //如果置零则输出 0
    else mealy_out <= (state == S010 && in ==1) ? 1'b1 : 1'b0; //检查是
否满足状态要求 并输出
end

endmodule
```

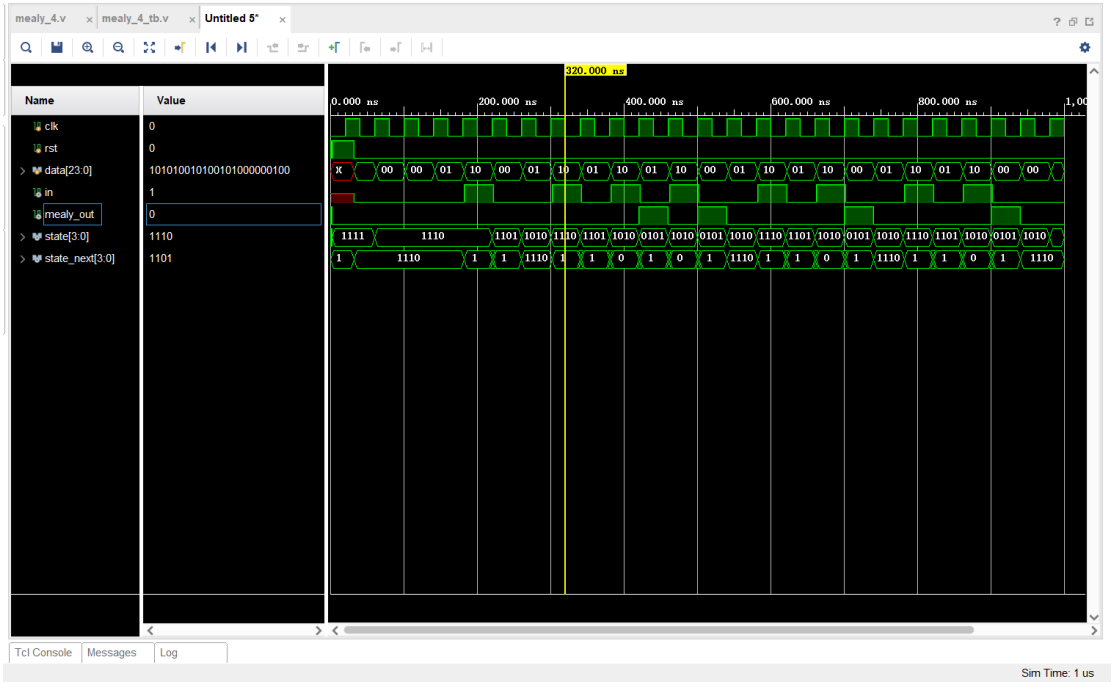
仿真文件：

```
//mealy_4_tb
module mealy_4_tb();
    reg clk, rst;   //时钟 置零
    reg [23:0] data;    //数据输入
    wire in,mealy_out;  //输入 输出
    wire [3:0] state,state_next;    //状态输出

    assign in = data[23];       //最新的哪个 in

    //关联
    mealy_4 M4(.in(in),.clk(clk),.rst(rst),.mealy_out(mealy_out),.state
(state),.state_next(state_next));
    //初始化
    initial begin
        clk = 1'b0;
        rst = 1'b0;
        #2 rst = 1'b1;
        #30 rst = 1'b0;
```

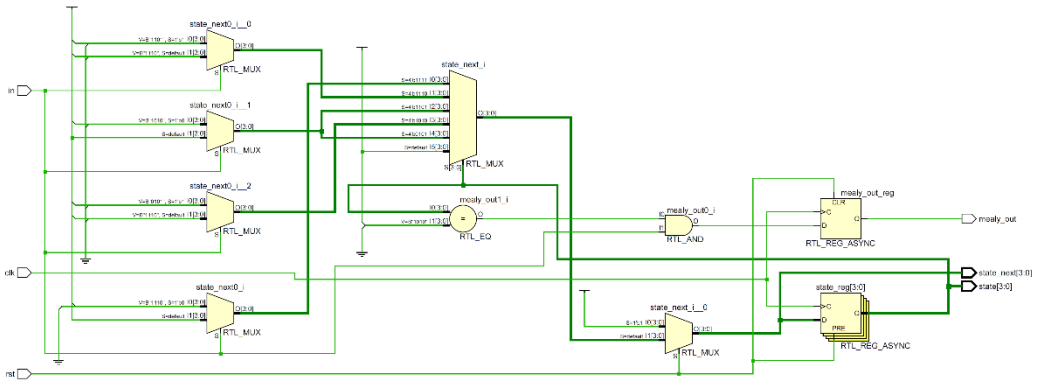
```
        data = 20'b1001_0101_0010_1001_0100;
        #20000 $finish;
    end

    //周期变化
    always #20 clk = ~clk;
    always @(posedge clk) #2 data = {data[22:0], data[23]};

endmodule
```

五、实验结果

仿真图：

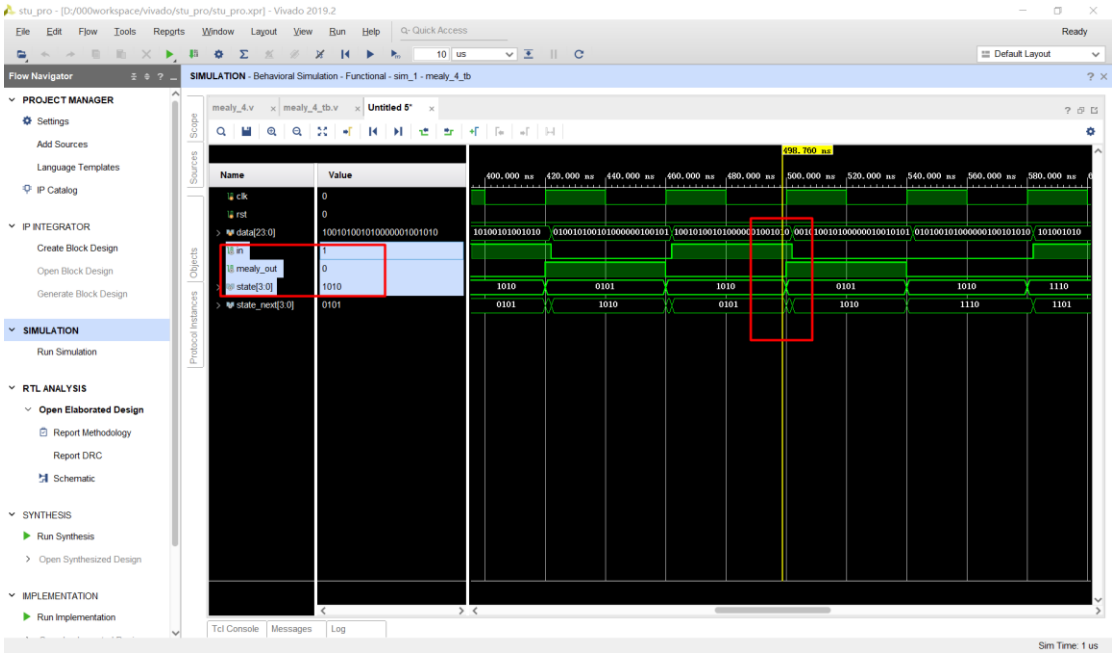


生成原理图：

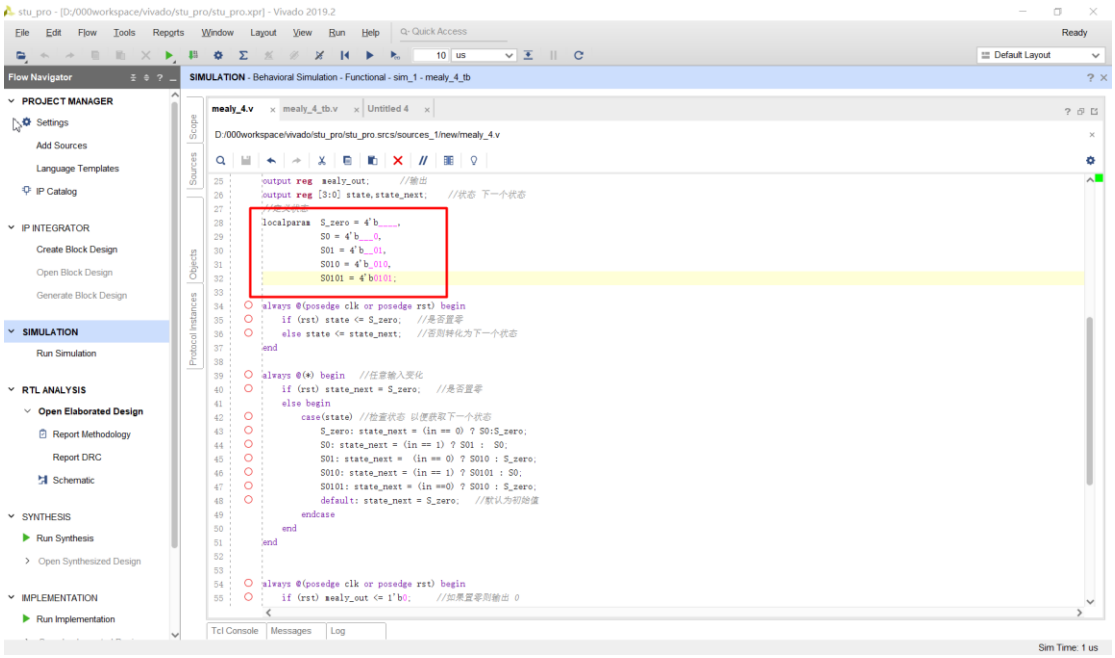


六、实验调试问题与解决办法

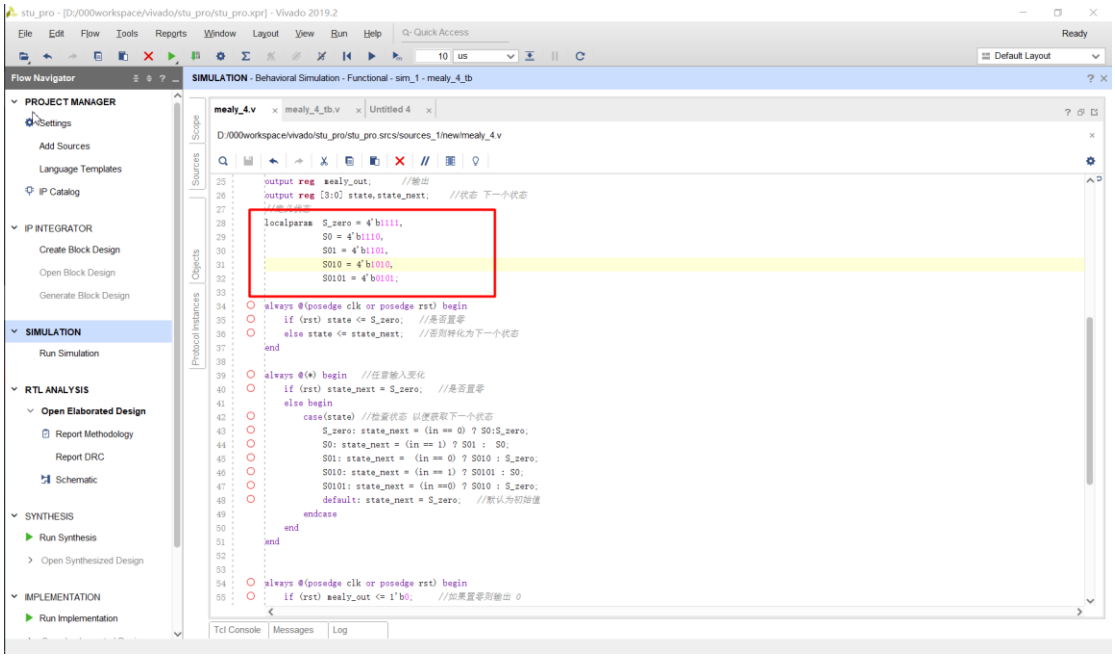
结果正确验证：



问题：数据变量错误，状态表示错误，下图为找到的错误。



错误解决办法：更改状态表达方法



实验小结

小结：

本次实验分别做了 8-3 编码器、8 选 1 数据选择器、环形右移寄存器、0101mealy 序列检测状态机。

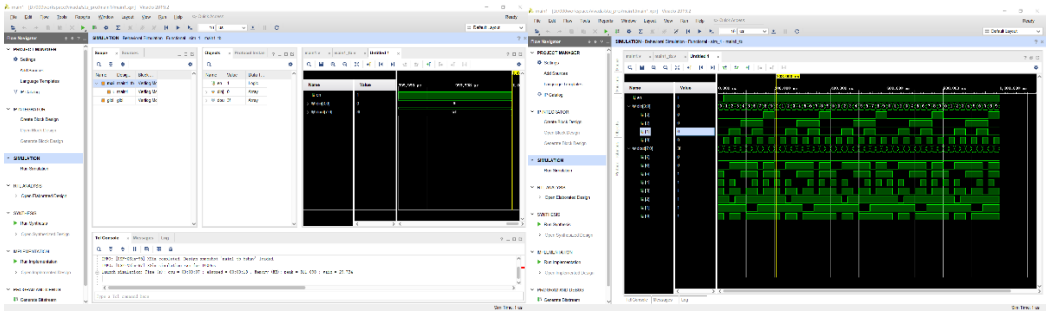
8-3 编码器、8 选 1 数据选择器以及环形右移寄存器与书本上并无多大区别。而序列检测则使用了三段式即三个 always 块，一个 always 模块采用同步时序描述状态转移；一个 always 采用组合逻辑判断状态转移条件，描述状态转移规律；第三个 always 块使用同步时序描述状态输出，寄存器输出。三段式与二段式相比，关键在于根据状态转移规律，在上一状态根据输入条件判断出当前状态的输出，从而在不插入额外时钟节拍的前提下，实现了寄存器输出。

心得体会：

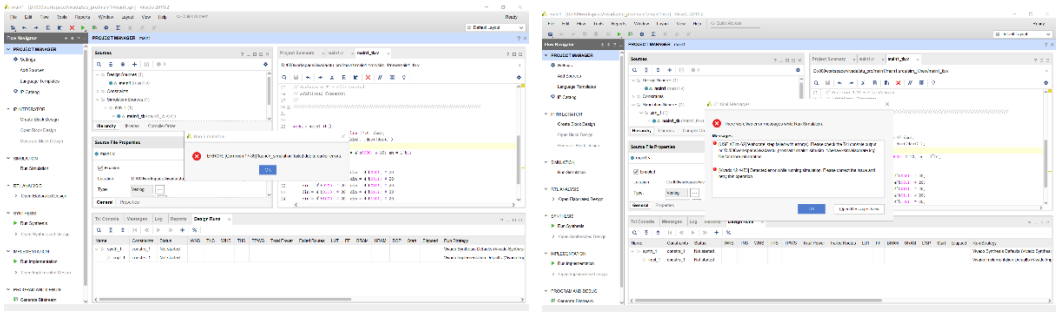
要写代码前必须对具体的硬件有一个比较清晰的概念，Verilog 的自顶向下设计方法就是从行为建模开始的，想一次完成可综合代码就太夸张了。Verilog 很基础的学习起来感觉不是很难，因为学过 c++有点基础。而且作为未来计算机领域人士，Verilog 是值得研究与学习的。实验过程体会了从无到有，感受了从报错到仿真成功得到美好的图片的过程，是自我成长中又一道美好的回忆。

初次接触 Verilog 记录

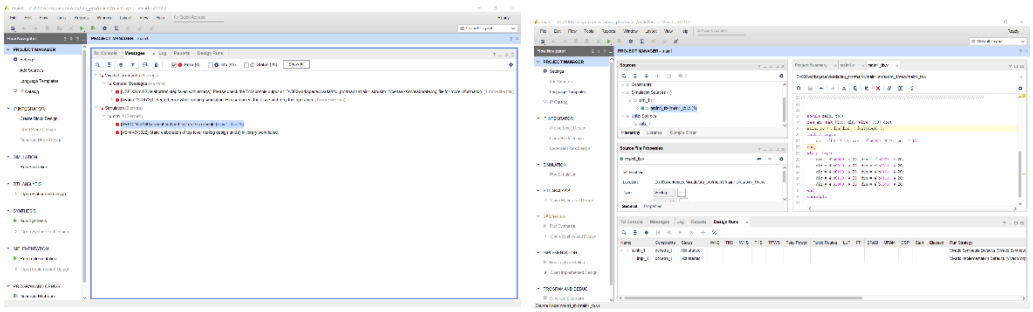
第一次运行成功：



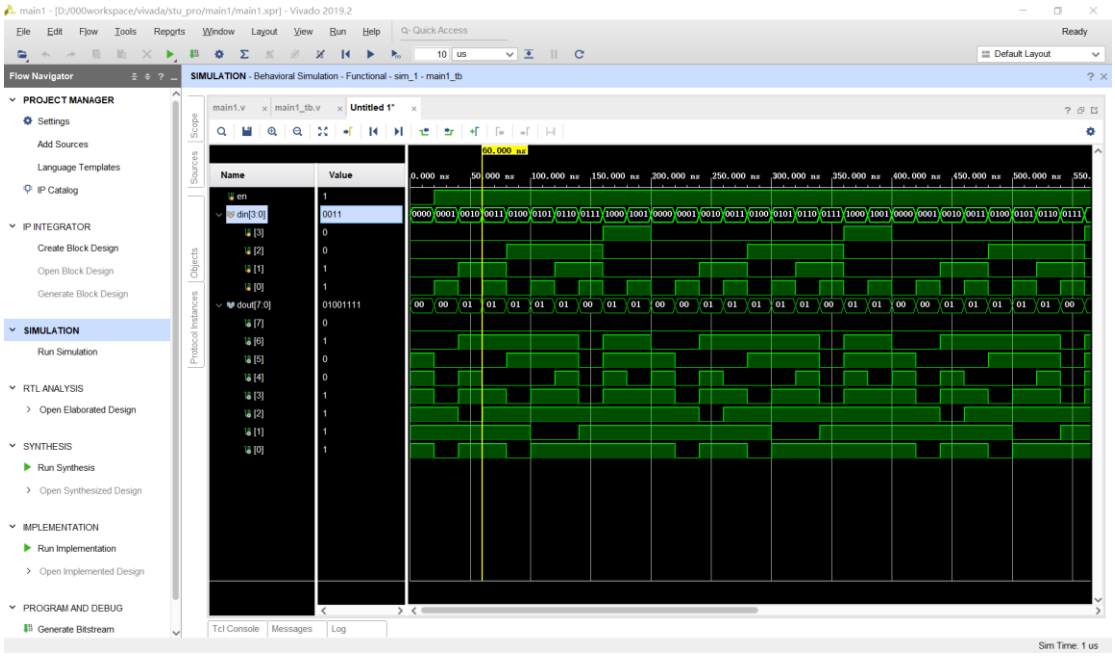
第一次出错：



第一次找到错误并修正:



第一次结束:


main1 - [D:/000workspace/vivada/stu_pro/main1/main1.xpr] - Vivado 2019.2
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Ready
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
SIMULATION - Behavioral Simulation - Functional - sim_1 - main1_tb
main1.v
main1_tb.v
Untitled 1*
Name
Value
en
din[3:0]
dout[7:0]
01001111
Tcl Console
Messages
Log
Sim Time: 1 us