**实验报告**

姓名： 专业： 电子科学与技术 学号：

课程名称： 数字系统 任课老师： 叶德信

实验名称： Dedicate Microprocessor Lab 实验日期： 2020.5.18

**1 实验目的和要求**

To learn how to implement a dedicated microprocessor.

学习如何做一个专用处理器。

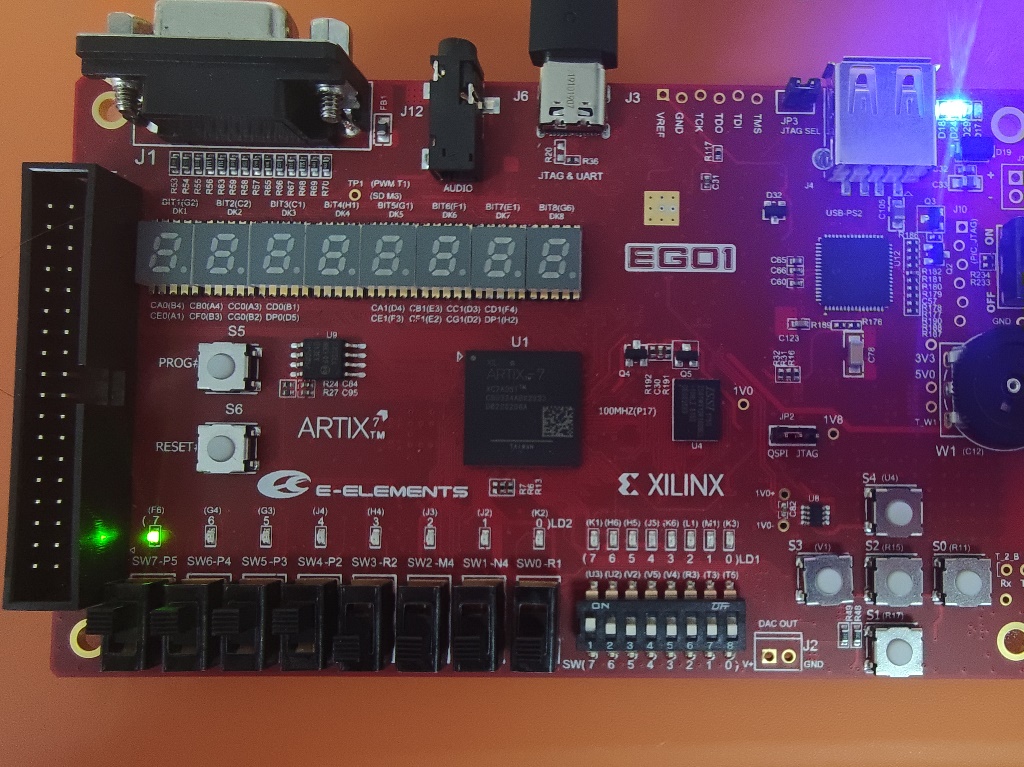
**2 实验原理**

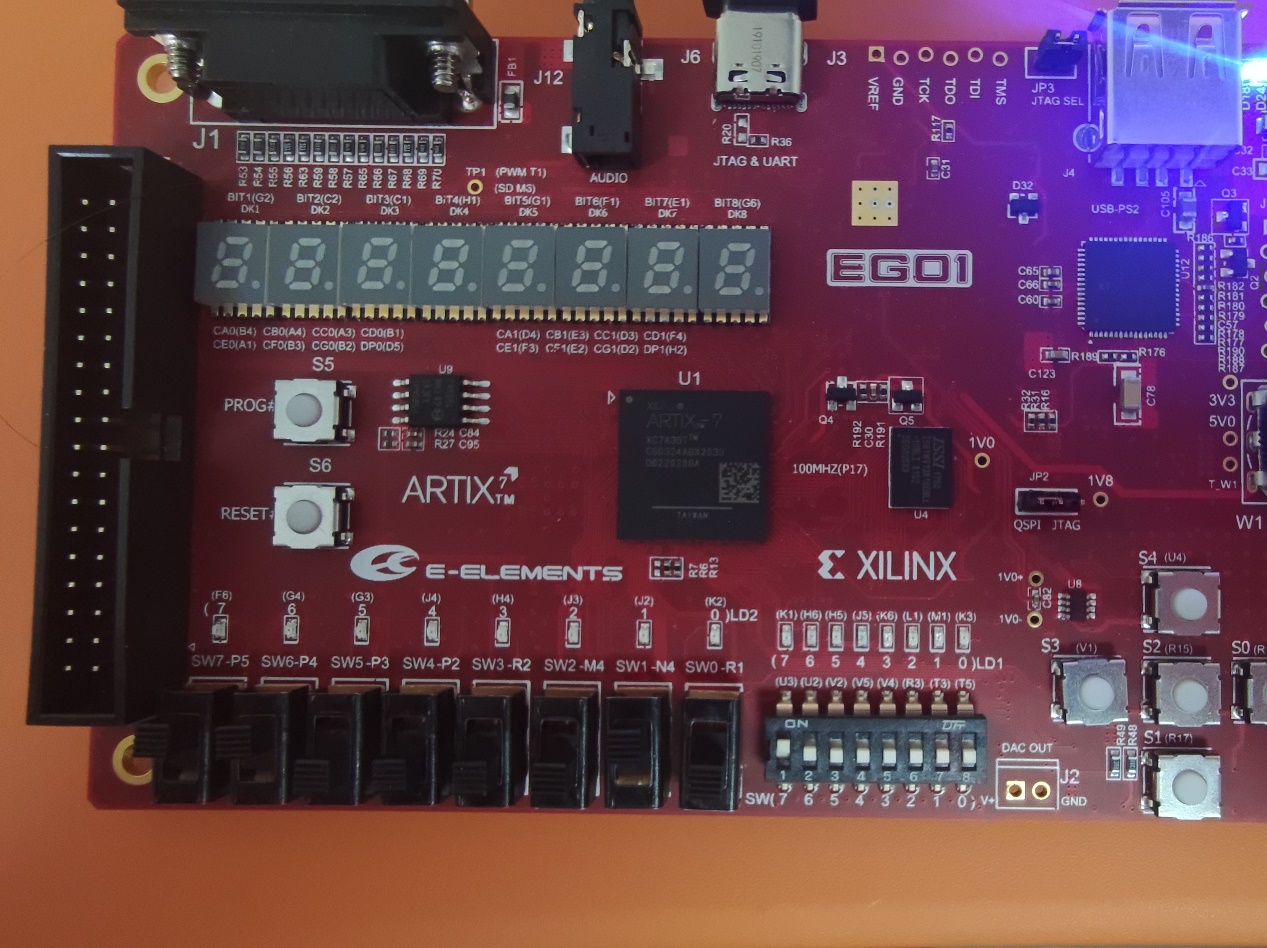
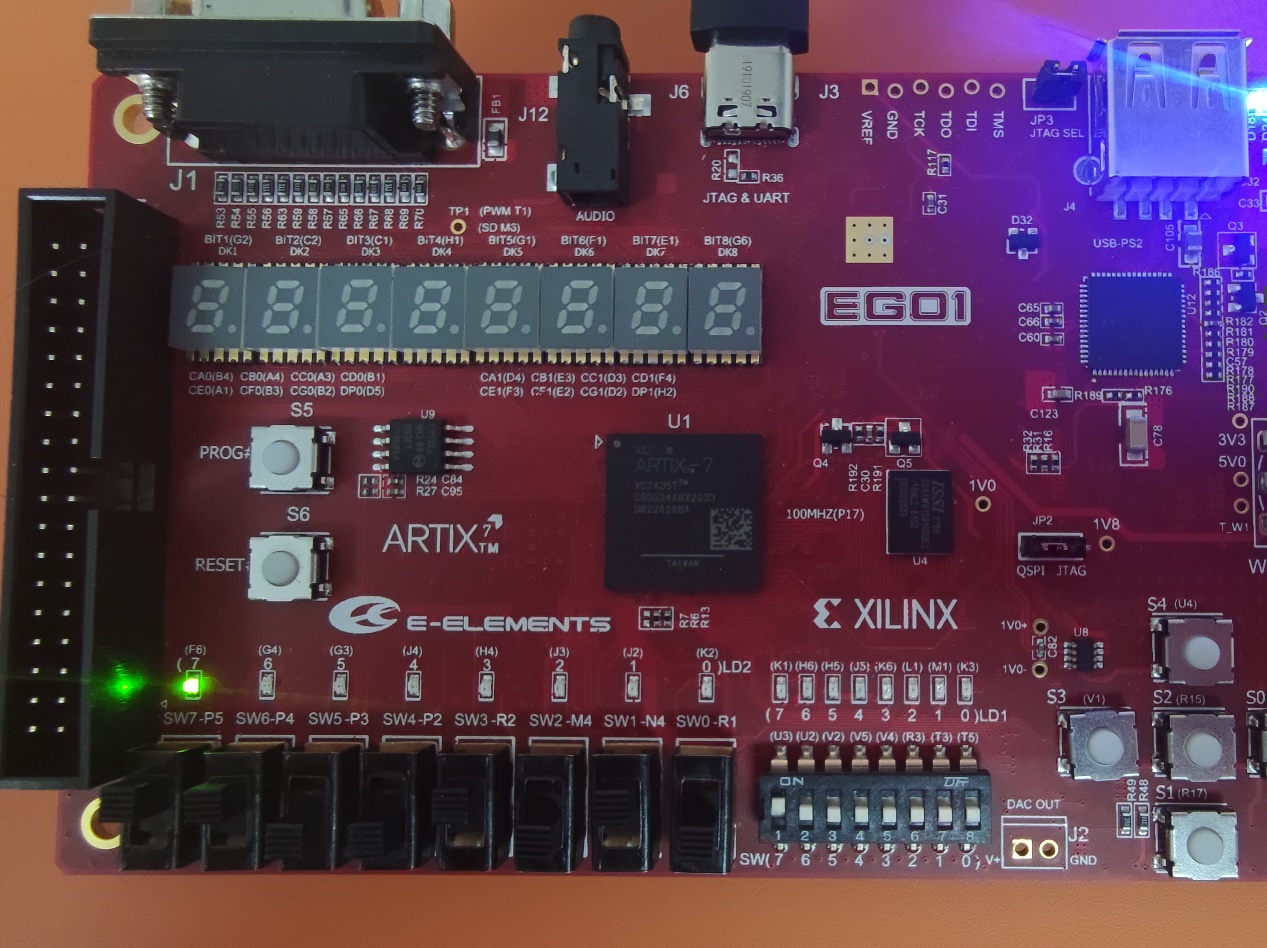
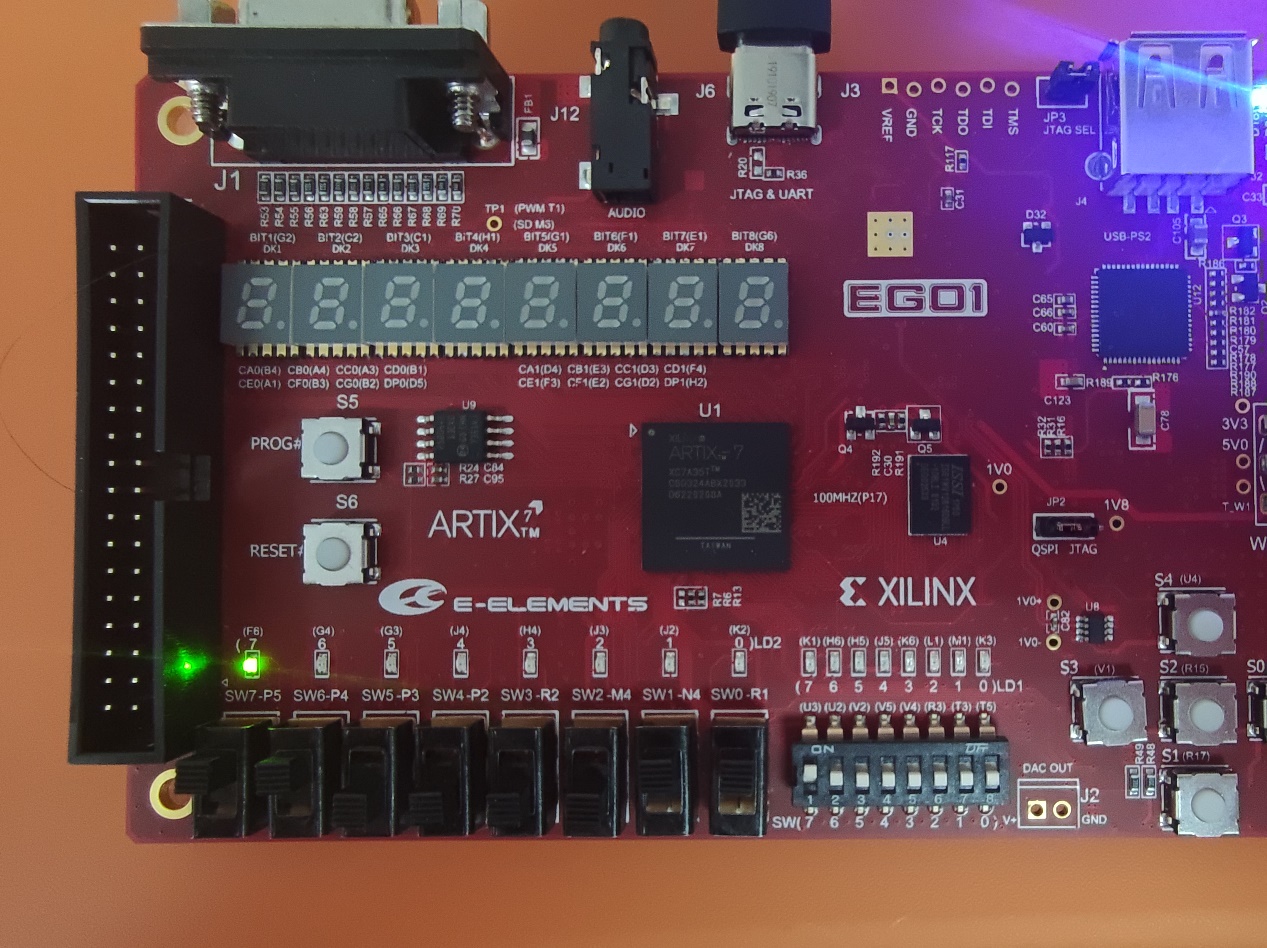
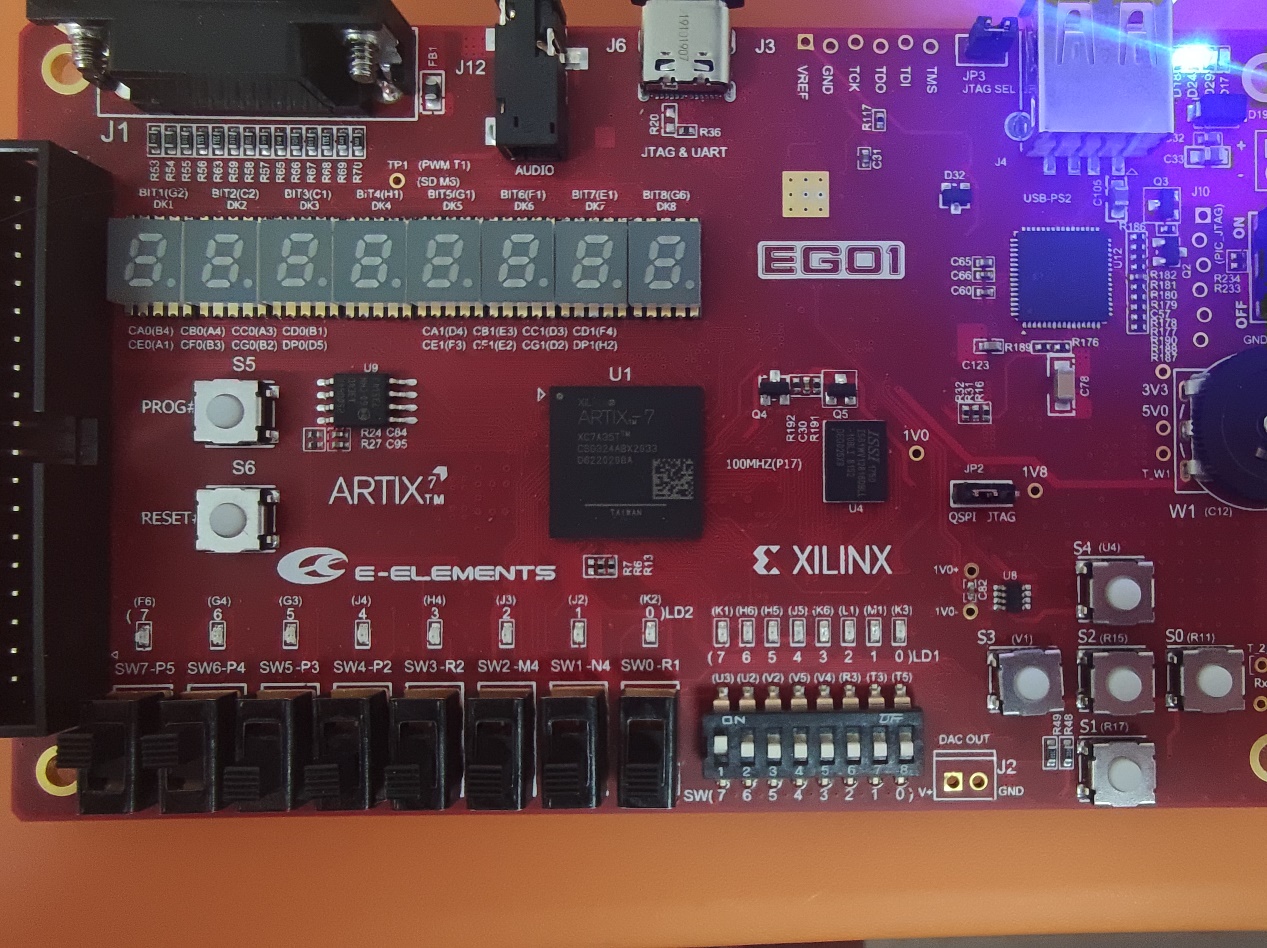
通过状态机和相应的数据通路，实现一个特定功能

**3 实验内容**

根据功能画状态图和数据通路，写控制字、数据通路和状态机代码。

**4 实验结果和分析**

把SW开关作为输入数据，灯的亮灭作为输出。



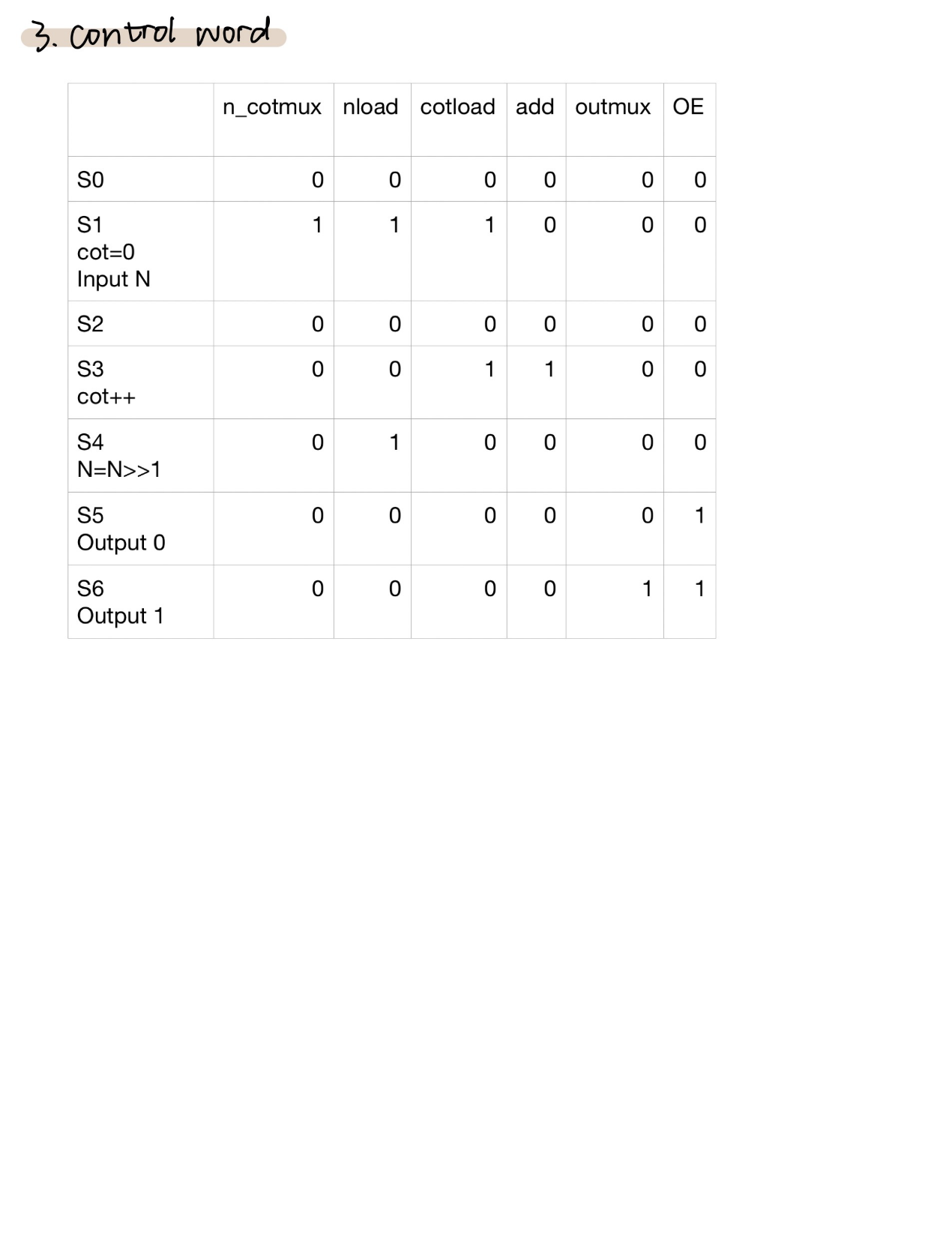
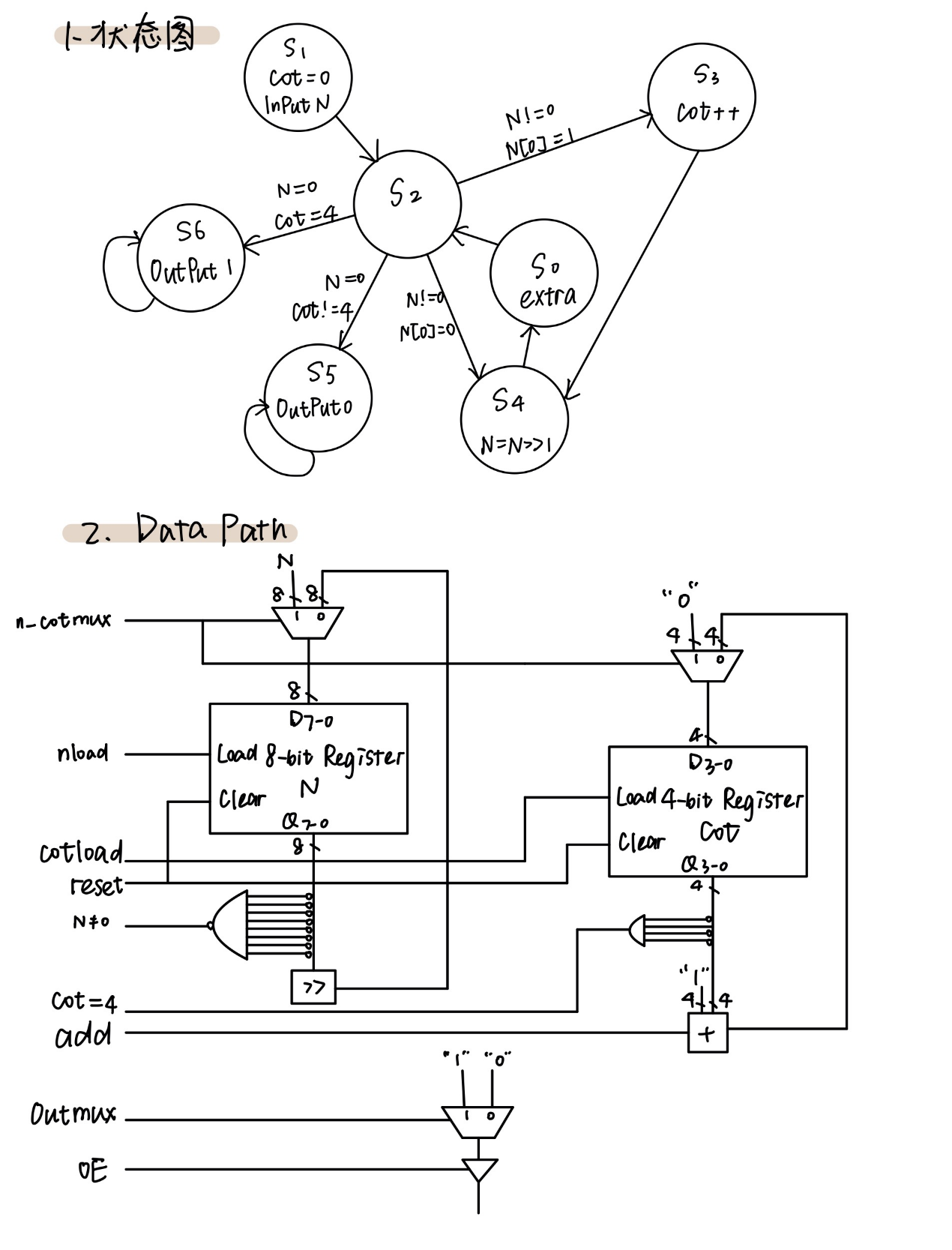
**5 实验结论和讨论**

实验结论： 通过SW开关来充当输入数据，输入完成后按reset，系统开始工作并输出结果。

遇到的问题和解决方案：

1. Q：数据通路中发现控制字不足以完成功能？ A：增加了一个控制字做加法器的判断。
2. Q：模拟时数据的最高位始终为Z？A：发现数据通路中的wire位数设置错误。

**附件：1、状态图、数据通路和控制字**

**2、源代码**

module **main**(

input [7:0]N,

input clk,resetin,

output LED

);

wire [7:0]N\_out;

wire [2:0]cot;

wire reset,n\_cotmux,outmux,nload,cotload,add,OE;

button U1(clk, resetin, reset);

FSM U2(N\_out, cot, clk, reset, n\_cotmux, nload, cotload, add, outmux, OE);

datapath U3(n\_cotmux, nload, cotload, add, outmux, OE, clk, reset, N, N\_out, cot, LED);

endmodule

module **button**(

input clk,button\_in,

output reg light

);

reg button,btemp;

always@(posedge clk)

{button,btemp} <= {btemp,button\_in};

wire bpressed ;

debounce d1(.clk(clk),. bounce(button),. signal(bpressed));

reg old\_bpressed;

always@(posedge clk)

begin

if(old\_bpressed == 0 && bpressed == 1) light <= 1;

else light <= 0;

old\_bpressed <= bpressed;

end

module **debounce**(

input clk, bounce,

output reg signal

);

parameter NDELAY = 650000;

reg [19:0] count;

reg xnew;

always@(posedge clk)

begin

if(bounce!= xnew) begin

xnew <= bounce; count <= 0;

end

else if(count == NDELAY) signal <= xnew;

else count <= count + 1;

end

endmodule

module **FSM**(

input [7:0]N,

input [2:0]cot,

input clk, reset,

output reg n\_cotmux, nload, cotload, add, outmux, OE

);

parameter S0=3'b000,S1=3'b001,S2=3'b010,S3=3'b011,S4=3'b100,S5=3'b101,S6=3'b110;

reg [2:0]state,next\_state;

always@(posedge clk or posedge reset)begin

if(reset) state<=S1;

else state<=next\_state;

end

always@(posedge clk)begin

if(state==S0)begin

n\_cotmux<=0;nload<=0;cotload<=0;add<=0;outmux<=0;OE<=0;

end

else if(state==S1)begin

n\_cotmux<=1;nload<=1;cotload<=1;add<=0;outmux<=0;OE<=0;

end

else if(state==S2)begin

n\_cotmux<=0;nload<=0;cotload<=0;add<=0;outmux<=0;OE<=0;

end

else if(state==S3)begin

n\_cotmux<=0;nload<=0;cotload<=1;add<=1;outmux<=0;OE<=0;

end

else if(state==S4)begin

n\_cotmux<=0;nload<=1;cotload<=0;add<=0;outmux<=0;OE<=0;

end

else if(state==S5)begin

n\_cotmux<=0;nload<=0;cotload<=0;add<=0;outmux<=0;OE<=1;

end

else if(state==S6)begin

n\_cotmux<=0;nload<=0;cotload<=0;add<=0;outmux<=1;OE<=1;

end

end

always@(\*)begin

if(state==S1) next\_state=S2;

else if(state==S2)begin

if(N[0]==1&&N!=0) next\_state=S3;

else if(N[0]==0&&N!=0) next\_state=S4;

else if(N==0&&cot!=4) next\_state=S5;

else if(N==0&&cot==4) next\_state=S6;

end

else if(state==S3) next\_state=S4;

else if(state==S4) next\_state=S0;

else if(state==S0) next\_state=S2;

else if(state==S5) next\_state=S5;

else if(state==S6) next\_state=S6;

end

module **datapath**(

input n\_cotmux,nload,cotload,add,outmux,OE,

input clk,reset,

input [7:0]N,

output [7:0]N\_out,

output [2:0]cot,

output light

);

wire [7:0] m1\_to\_reg1,reg1\_to\_right,right\_to\_m1;

mux2\_8bits m1(N, right\_to\_m1, n\_cotmux, m1\_to\_reg1);

reg\_8bits reg1(m1\_to\_reg1, clk, reset, nload, reg1\_to\_right);

right\_move right(reg1\_to\_right, right\_to\_m1);

wire [3:0] m2\_to\_reg2 , reg2\_to\_add1 , add1\_to\_m2 ;

mux2\_4bits m2(0, add1\_to\_m2, n\_cotmux, m2\_to\_reg2);

reg\_4bits reg2(m2\_to\_reg2, clk, reset, cotload, reg2\_to\_add1);

add add1(1, reg2\_to\_add1, add, add1\_to\_m2);

wire m3\_to\_gate;

mux2\_1bit m3(1, 0, outmux, m3\_to\_gate);

tri\_gate gate(m3\_to\_gate, OE, light);

assign cot = reg2\_to\_add1;

assign N\_out = reg1\_to\_right;

endmodule

module **mux2\_8bits**(

input [7:0] ina,inb,

input sel,

output reg [7:0] outa

);

always@(\*)begin

if(sel == 1) outa = ina;

else outa = inb;

end

endmodule

module **mux2\_4bits**(

input [3:0] ina,inb,

input sel,

output reg [3:0] outa

);

always@(\*)begin

if(sel == 1) outa = ina;

else outa = inb;

end

endmodule

module **mux2\_1bit**(

input ina,inb,

input sel,

output reg outa

);

always@(\*)begin

if(sel == 1) outa = ina;

else outa = inb;

end

endmodule

module **reg\_8bits**(

input [7:0] ina,

input clk, reset, nload,

output reg [7:0] outa

);

always@(posedge clk or posedge reset)begin

if(reset) outa <= 8'b00000000;

else if(nload == 1) outa <= ina;

end

endmodule

module **reg\_4bits**(

input [3:0] ina,

input clk, reset, cotload,

output reg [3:0] outa

);

always@(posedge clk or posedge reset)begin

if(reset) outa <= 4'b0000;

else if(cotload == 1) outa <= ina;

end

endmodule

module **right\_move**(

input [7:0] ina,

output reg [7:0] outa

);

always@(\*)begin

outa <= { 0, ina[7:1] };

end

endmodule

module **add**(

input [3:0] ina, inb,

input sel,

output reg [4:0] outa

);

always@(\*)begin

if(sel == 1) outa <= ina + inb;

end

endmodule

module **tri\_gate**(

input ina, OE,

output reg outa

);

always@(\*)begin

if(OE == 1) outa <= ina;

end

endmodule

**3、仿真代码及结果**

module sim1(

);

reg [7:0] n;

reg clk,reset;

wire light;

main sim(n,clk,reset,light);

initial

begin

reset = 0;

n = 8'b00001111;

end

initial

begin

clk = 0;

forever

#1

clk = ~clk;

end

always

begin

#30

reset = 1;

#30

reset = 0;

#1000

n = 8'b00000111;

reset = 1;

#30

reset = 0;

#1000

n = 8'b00110011;

reset = 1;

#30

reset = 0;

end

endmodule