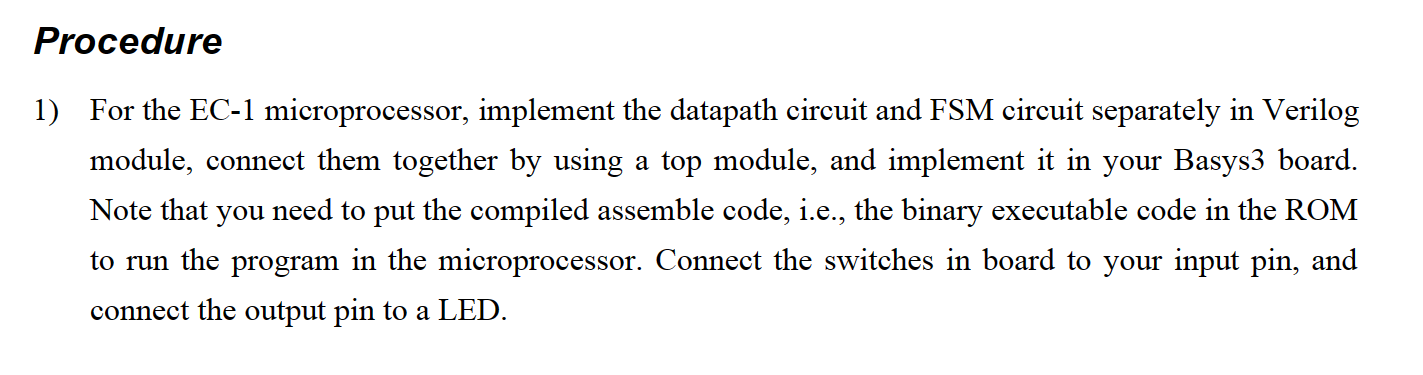
**1 实验目的和要求**

Goal To verify the design of EC1.



**2 实验内容**

（分点阐述实验步骤）

1. 分别编写数据通路和控制单元Verilog代码，设计一个顶层代码连接两模块。
2. 在rom中编写汇报代码，上板测试。

**3 源代码与分析**

**各模块代码如下：**

**递增器：**

module increment (

    input [3:0] A,

    output [3:0] F

);

    assign F = A + 1;

endmodule

**选择器：**

module mux21

#(parameter n=4)

(

    input S,

    input [n-1:0]A,

    input [n-1:0]B,

    output [n-1:0]Y

);

    assign Y = S ? A : B;

endmodule

**寄存器：**

module Register

#(parameter n=8)

(

 input clk,rst,load,

 input [n-1:0] D,

    output reg [n-1:0] Q

);

    always @(posedge clk or negedge rst ) begin

        if (rst==0) begin

            Q<=0;

        end

        else if (load==1) begin

            Q<=D;

        end

    end

endmodule

**ROM：**

module rom

#(parameter size = 4)

    (

    input E,

    input [size-1:0] addr,

    output [7:0] data

    );

reg [7:0] mem[0:size\*\*2-1];

    initial begin

        mem[0] <= 8'b01100000;

        mem[1] <= 8'b10000000;

        mem[2] <= 8'b10100000;

        mem[3] <= 8'b11000001;

        mem[4] <= 8'b11111111;

    end

    assign  data = E? mem[addr] : 8'bzzzzzzzz;

endmodule

**三态门：**

module tri\_buffer

    #(parameter n = 4)

    (input E,

    input [n-1:0] D,

    output[n-1:0] Y

    );

    assign Y = E ? D : {n{1'bz}} ;

endmodule

**数据通路：**

`include "decrement.v"

`include "increment.v"

`include "mux21.v"

`include "Register.v"

`include "rom\_16x8.v"

`include "tri\_buffer.v"

module dp (

    input clk,rst,

    input IRload,PCload,

    input JNZmux,INmux,

    input Aload,

    input OutE,

    input [7:0] Input,

    output [2:0]IR75,

    output Anot0,

    output [7:0] Output

);

    wire[7:0] dp\_ROMQ,dp\_IR;

    wire[3:0] dp\_increment,dp\_PC,dp\_JNZmux;

    wire[7:0] dp\_INmux,dp\_A,dp\_decrement;

    Register #(8) regIR (.clk(clk),.rst(rst),.load(IRload),.D(dp\_ROMQ),.Q(dp\_IR));

    Register #(4) regPC (.clk(clk),.rst(rst),.load(PCload),.D(dp\_JNZmux),.Q(dp\_PC));

    Register #(8) regA (.clk(clk),.rst(rst),.load(Aload),.D(dp\_INmux),.Q(dp\_A));

    mux21 #(8) muxIN (.A(Input),.B(dp\_decrement),.S(INmux),.Y(dp\_INmux));

    mux21 #(4) muxJNZ (.A(dp\_IR[3:0]),.B(dp\_increment),.S(JNZmux),.Y(dp\_JNZmux));

    increment inc (.A(dp\_PC),.F(dp\_increment));

    decrement dec (.A(dp\_A),.F(dp\_decrement));

    rom rom (.E(1'b1),.addr(dp\_PC),.data(dp\_ROMQ));

    tri\_buffer #(8) buffer (.E(OutE),.D(dp\_A),.Y(Output));

    assign IR75 = dp\_IR[7:5];

    assign Anot0 = (dp\_A!=0)? 1'b1 : 1'b0;

endmodule

**控制单元：**

module cu (

    input clk,

    input rst,

    input [2:0] IR75,

    input Anot0,

    output reg IRload,

    output reg PCload,

    output reg JNZmux,

    output reg Halt\_out,

    output reg OutE,

    output reg Aload,

    output reg INmux

);

    parameter Fetch = 3'b000,Decode = 3'b001,Input = 3'b011,Output = 3'b100,Dec=3'b101,JNZ=3'b110,Halt=3'b111;

    reg[2:0] state,next\_state;

    always @(posedge clk or negedge rst  ) begin

        if(!rst) begin

            state <= Fetch;

        end

        else begin

            state <= next\_state;

        end

    end

    always @(posedge clk or negedge rst) begin

        if (!rst) begin

            IRload<=1;

            PCload<=1;

            JNZmux<=0;

            Halt\_out<=0;

            OutE<=0;

            Aload<=0;

            INmux<=0;

        end

        else begin case (next\_state)

            Fetch: begin

                IRload<=1;

                PCload<=1;

                JNZmux<=0;

                Halt\_out<=0;

                OutE<=0;

                Aload<=0;

                INmux<=0;

            end

            Decode: begin

                IRload<=0;

                PCload<=0;

                JNZmux<=0;

                Halt\_out<=0;

                OutE<=0;

                Aload<=0;

                INmux<=0;

            end

            Input: begin

                IRload<=0;

                PCload<=0;

                JNZmux<=0;

                Halt\_out<=0;

                OutE<=0;

                Aload<=1;

                INmux<=1;

            end

            Output: begin

                IRload<=0;

                PCload<=0;

                JNZmux<=0;

                Halt\_out<=0;

                OutE<=1;

                Aload<=0;

                INmux<=0;

            end

            Dec: begin

                IRload<=0;

                PCload<=0;

                JNZmux<=0;

                Halt\_out<=0;

                OutE<=0;

                Aload<=1;

                INmux<=0;

            end

            JNZ: begin

                IRload<=0;

                PCload<=(Anot0==1)?1:0;

                JNZmux<=1;

                Halt\_out<=0;

                OutE<=0;

                Aload<=0;

                INmux<=0;

            end

            Halt: begin

                IRload<=0;

                PCload<=0;

                JNZmux<=0;

                Halt\_out<=1;

                OutE<=0;

                Aload<=0;

                INmux<=0;

            end

            default: begin

                IRload<=0;

                PCload<=0;

                JNZmux<=0;

                Halt\_out<=0;

                OutE<=0;

                Aload<=0;

                INmux<=0;

            end

        endcase

        end

    end

    always @(state or IR75 ) begin

        case (state)

        Fetch: begin

            next\_state=Decode;

        end

        Decode: begin

            if (IR75==3'b011) next\_state=Input;

            else if (IR75==3'b100) next\_state=Output;

            else if (IR75==3'b101) next\_state=Dec;

            else if (IR75==3'b110) next\_state=JNZ;

            else if (IR75==3'b111) next\_state=Halt;

            else next\_state=Fetch;

        end

        Input: begin

           next\_state=Fetch;

        end

        Output: begin

            next\_state=Fetch;

        end

        Dec: begin

            next\_state=Fetch;

        end

        JNZ: begin

            next\_state=Fetch;

        end

        Halt: begin

            next\_state=Halt;

        end

        default: begin

            next\_state=Fetch;

        end

        endcase

    end

endmodule

**顶层模块：**

`include "dp.v"

`include "cu.v"

module top (

    input clk,rst,

    input [7:0] A,

    output [7:0] Output,

    output Halt\_out

);

    wire IRload,PCload,JNZmux,OutE,Aload,INmux;

    wire [2:0] IR75;

    wire Anot0;

    dp dp (

        .clk(clk),

        .rst(rst),

        .IRload(IRload),

        .PCload(PCload),

        .JNZmux(JNZmux),

        .OutE(OutE),

        .Aload(Aload),

        .INmux(INmux),

        .Input(A),

        .IR75(IR75),

        .Anot0(Anot0),

        .Output(Output));

    cu cu (

        .clk(clk),

        .rst(rst),

        .IR75(IR75),

        .Anot0(Anot0),

        .IRload(IRload),

        .PCload(PCload),

        .JNZmux(JNZmux),

        .Halt\_out(Halt\_out),

        .OutE(OutE),

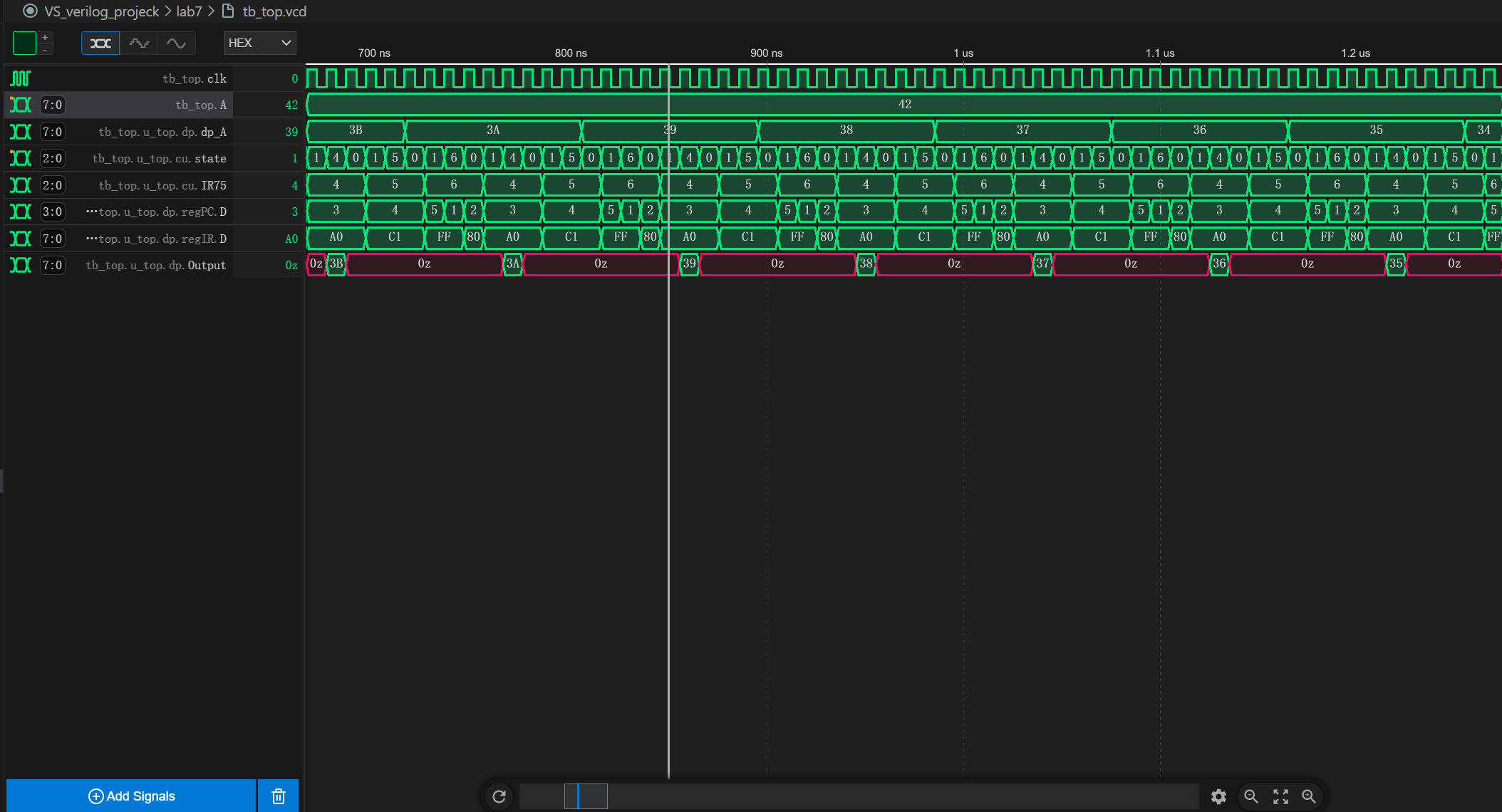
        .Aload(Aload),

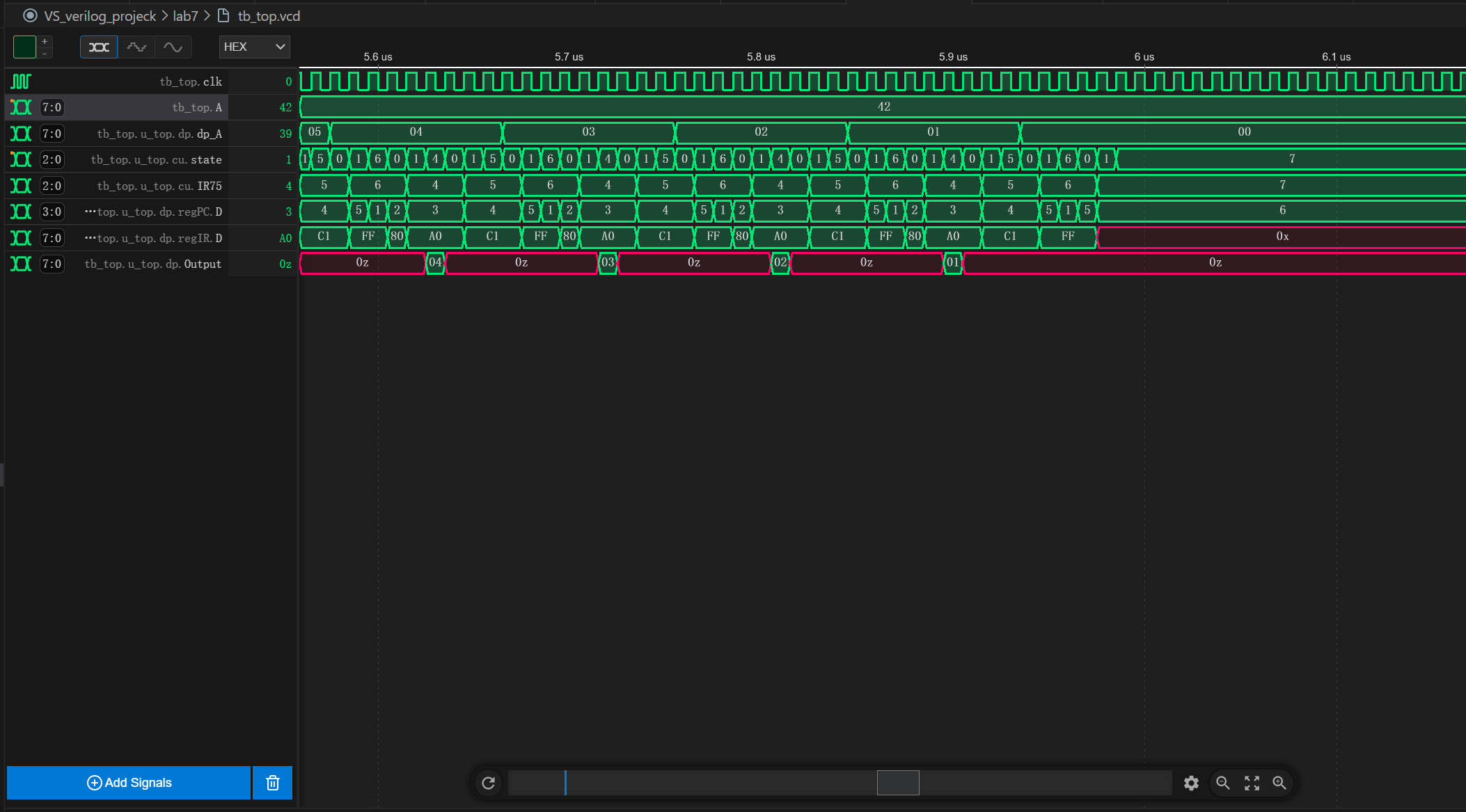
        .INmux(INmux));

endmodule

**4 实验结果和分析**

仿真结果如下，成功实现了rom中的汇编代码功能。

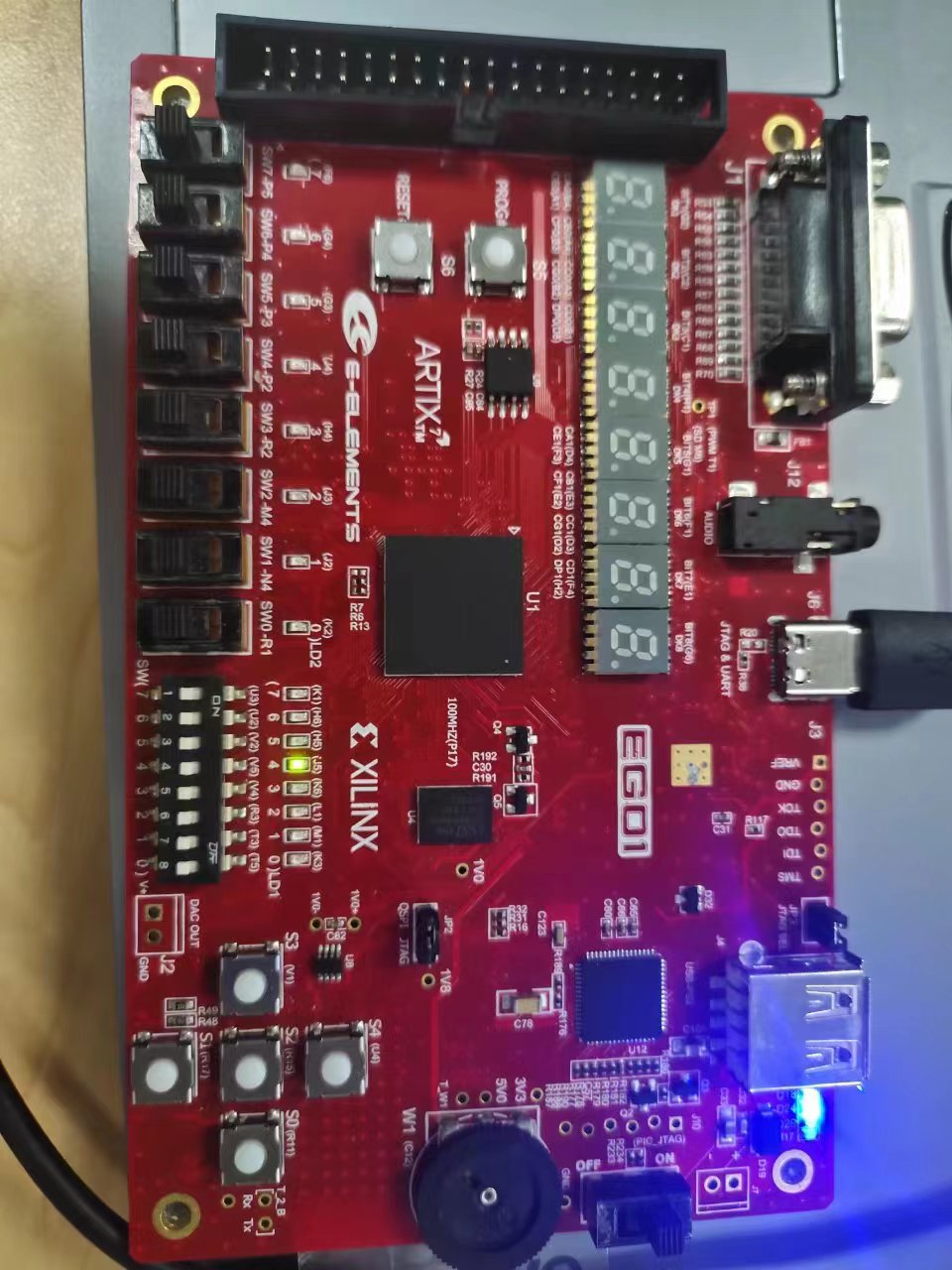




测试效果如下功能正常

将八个输入约束到八个拨码开关上，输出约束到八个led灯上，输出Halt\_out约束到LEDJ5

上板测试。结果如下



拨码开个输入后，按下reset按钮，八个led亮起代表信号开始递减。

8个Led全灭后，ledJ5亮起，表示程序结束。

如图所示功能正常。

**5 实验结论**

（基于实验结果和分析，得出结论）

成功使用Verilog代码，实现了lab所要求的功能。