Computer Organization and Architecture Course Design

Southeast University
School of Information Science and Technology

Author: WU YX

Student Number: 040193xx

Project Name: A Parallel Output Controller(POC)

Date: March 23, 2022

Contents

1.Purpose

The purpose of this project is to design and simulate a parallel output controller (POC) which acts an interface between system bus and printer. The Xilinx Vivado is adopted for simulation.

2. Tasks

POC is one of the most common I/O modules, namely the parallel output controller. It plays the role of an interface between the computer system bus and the peripheral (such as a printer or other output devices).

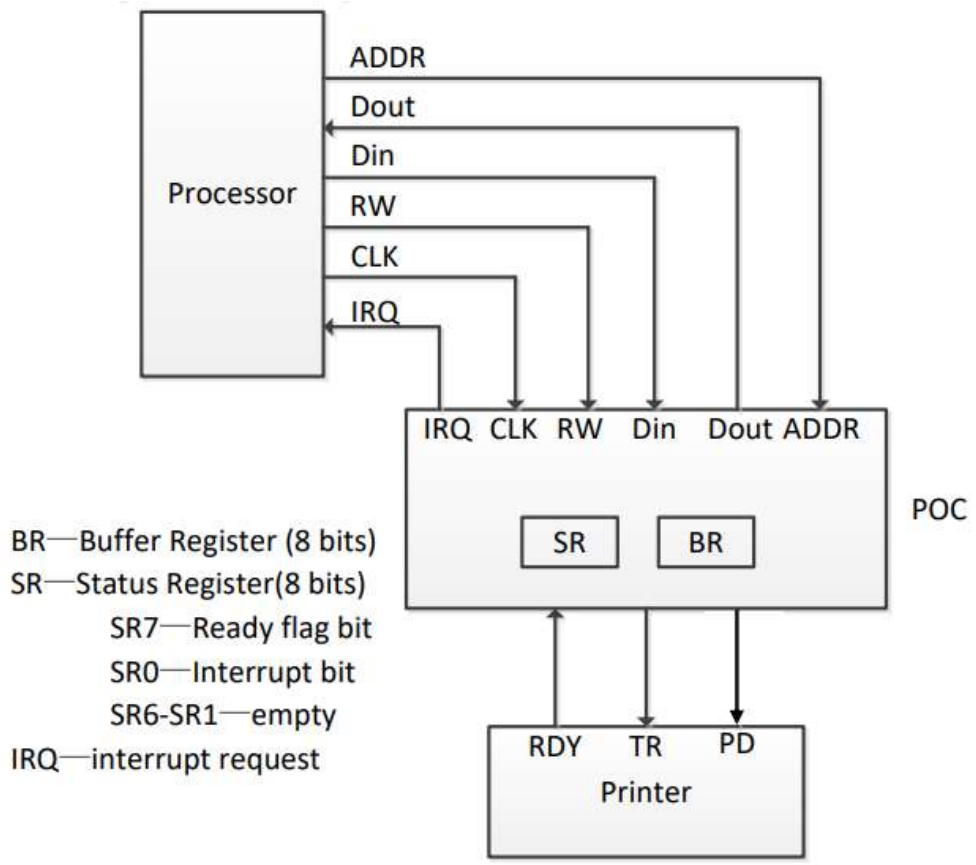


Figure 1 Printer Connection

Figure 1 shows the connecting of a printer to the system bus through the POC. The communication between POC and the printer is controlled by a "handshake" protocol illustrated in Figure 2.

The handshaking process is described as follows: When the printer is

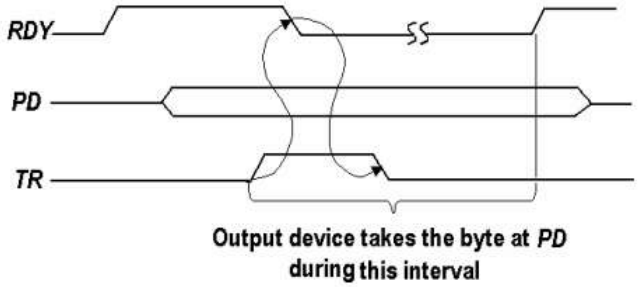


Figure 2 The handshake timing diagram between POC and the printer

ready to receive a character, it holds RDY=1.The POC then hold a character at PD (parallel data) port and generate a pulse at the port TR (transfer request). The POC will change TR to 0 when detecting printer has respond TR, i.e., RDY has been changed from 1 to 0. When detecting the effective TR signal, the printer will change RDY to 0, take the character at PD and hold the RDY at 0 until the character has beenprinted (e.g. 5 or 10ms), then set RDY=1 again when it is ready to receive the next character. (Suppose the printer has only a one character "buffer" register, so that each character must be printed before the next character is sent).

The buffer register BR is used to temporarily hold a character sent from the processor, which character will be transferred to the printer later. The status register SR is used for two control functions: SR7 serves as a ready flag to indicate POC is ready or not to receive a new character from the processor, and SR0 is used to enable the interrupt requests sent by POC. In in interrupt mode, If SR0=1, then POC will send an interrupt request signal to processor when it is ready to receive a character (i.e., when SR7=1). If SR0=0, then POC will not interrupt. The other bits of SR are not used and empty.

3. Design Analysis

3.1 Overall Design

To adopt modular design , the author designed four independent modules first, namely, Top, Processor, POC and Printer. In Top module, We

proviede the whole system with necessary inputs, including clk, modest, reset, and receives the printer output from the system by instantiating the three modules. The connections of all the modules is shown in figure3.

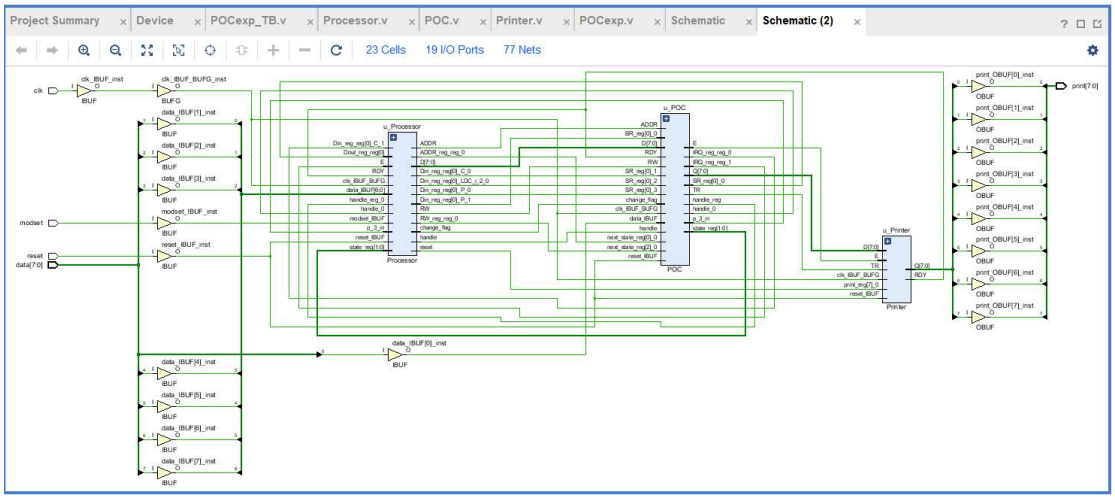


Figure 3 Top Module schematic

3.2 Processor Design

```
23  module Processor(
24      output ADDR,//控制SR还是BR，1是BR
25      input [7:0]Dout,
26      output [7:0]Din,
27      output RW,//控制读写，1是cpu向外写
28      input clk,
29      input IRQ,//IRQ为0代表中断请求
30      input reset,
31      input modset,//1中断0查询
32      input [7:0]data
33
34      );
```

Signal Explanations:

Input

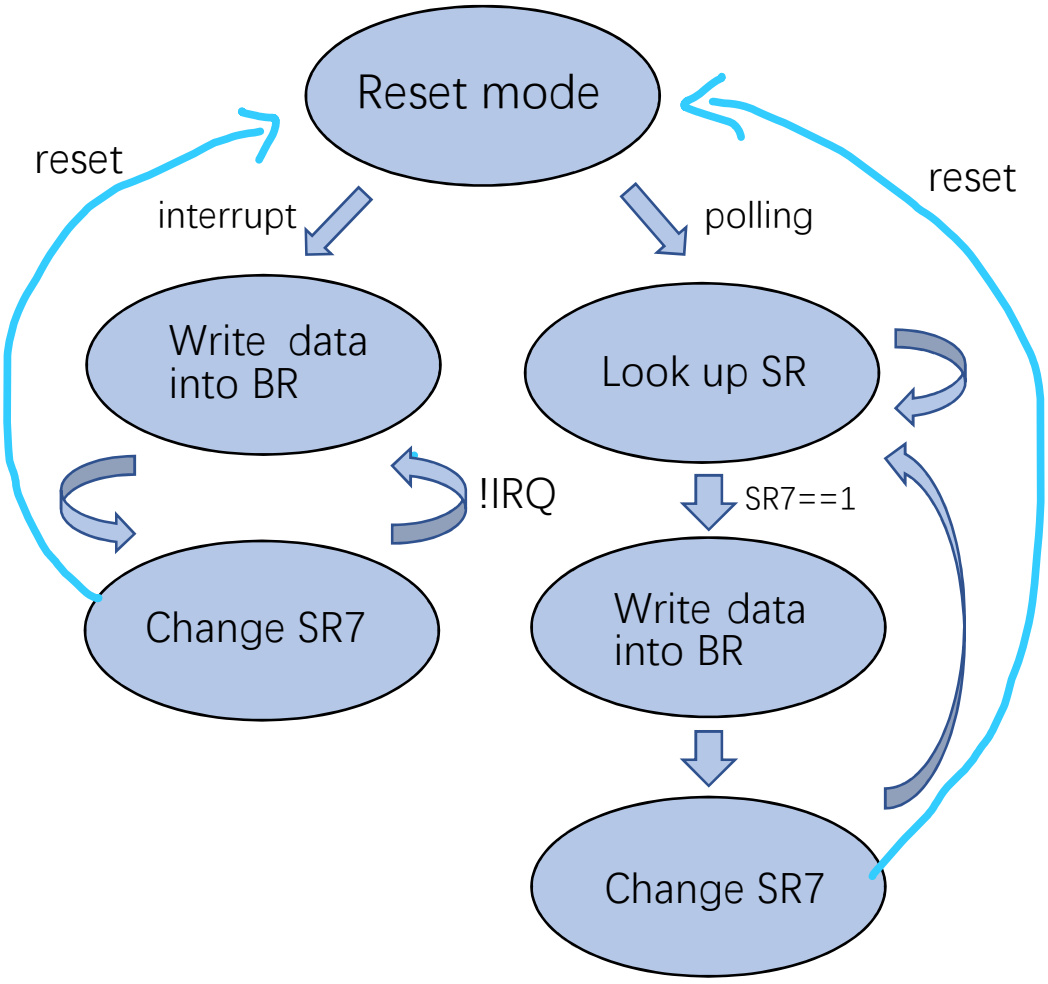
[7:0]Dout	Data which is read from POC
[7:0]data	Input data, waiting to be written to POC
clk	Clock signal
IRQ	Interrupt signal read from POC, 0 implies enabled
modset	Change modes from the outside, 1 for interrupt
reset	0 for restart, coordiantes with modset

Output	
ADDR	Address select control, 1 for BR, 0 for SR
RW	Read or write control, 1 for write into POC
[7:0]Din	Data to be written into POC

Besides, I've designed two internal registers to meet certain functions.

reg handle	a flag that indicates if the written operation has been done by CPU in one circle.
reg change_flag	1 clock time delay which helps the mod change

For the purpose of the design is to simulate a POC instead of a CPU, we use if-else branches to depict the logic instead of a FSM.



3.3 POC Design

```
23  module POC(
24      input clk,
25      output IRQ,
26      input RW,
27      input [7:0]Din,
28      output [7:0]Dout,
29      input ADDR,
30      input RDY,
31      input reset,
32      // input mod,//1中断0查询
33      output TR,
34      output [7:0]PD
35
```

Signal Explanations:

Input

[7:0]Din	Data from CPU.
RW	Input data, waiting to be written to POC
clk	Clock signal
RDY	Ready flag from Printer, 1 for ready
ADDR	Address select control, 1 for BR, 0 for SR
reset	0 for restart

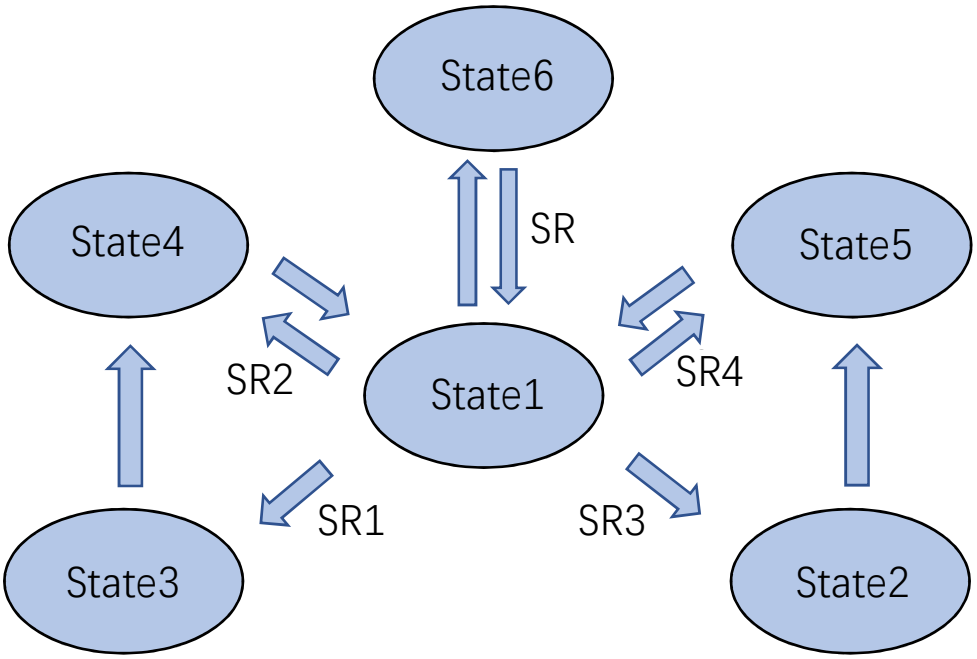
Output

TR	Impulse which activates Printer
IRQ	Interrupt flag, 0 for interrupt
[7:0]PD	Data to be sent to Printer
[7:0]Dout	Data to be sent to CPU

The simulation of POC adopts finite state machine.

parameter state1= 3'b000;	start
parameter state2= 3'b001;	Interrupt

parameter state3= 3'b010; Polling
parameter state4= 3'b011; handshake after polling
parameter state5= 3'b100; handshake after interrupt
parameter state6= 3'b101; refresh SR
SR1:10000000 SR2:00000000
SR3:10000001 SR4:00000001



Inside state2 and state3, if-else branches are used to communicate with CPU according to the content of RW and ADDR.
Inside state4 and state5, if RDY==1, POC will activate TR and send data to printer, SR7 is also reset to 1.

3.4 Printer Design

```
23  module Printer(
24      input clk,
25      input TR,
26      input [7:0]PD,
27      input reset,
28      output RDY,
29      output [7:0]PRINT
```

Signal Explanations:

Input

[7:0]PD	Data from POC
TR	If TR==1, start receiving data
clk	Clock signal
reset	0 for restart

Output

[7:0]PRINT	Output the received data
RDY	Ready flag for POC

In this module, logic is rather clear, it just needs to receive data and print. A waiting flag and a counter is set to simulate print delay.

4. Simulation

4.1 Input Waveform Design

```
55  begin
56      data=8'b10100101;//第一个data
57      reset=1;//先重开
58      modset=1;//先中断
59      #(JUMP);
60      reset=0;//边沿触发
61      #(JUMP);
62      reset=1;
63
64      #(PERIOD*20);
65      data=8'b11110000;//第2个data
66      #(PERIOD*20);
67      data=8'b00001111;//第3个data
68      #(PERIOD*20);
```

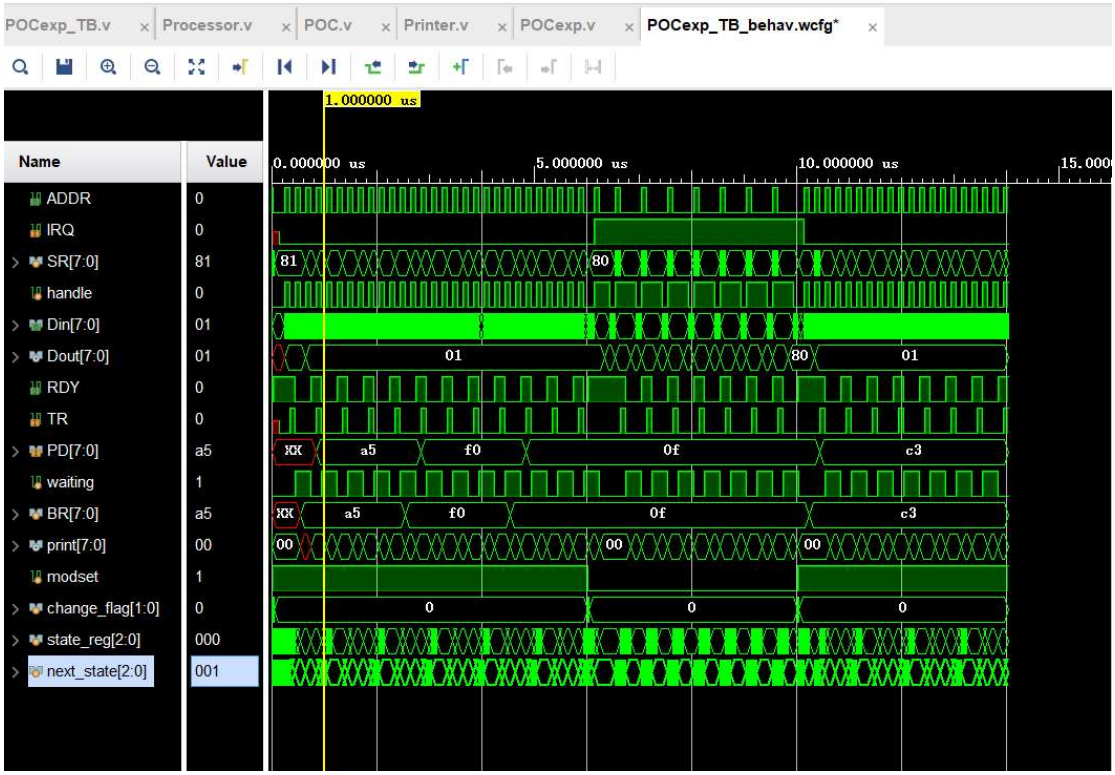
```
70      reset=0;
71      modset=0;
72      #(JUMP);
73      reset=1;
74      //#(PERIOD*20);
75     // data=8'b00110011;//第4个data
76      #(PERIOD*40);
77      data=8'b11000011;//第4data
78
79      reset=0;//切换
80      modset=1;
81      #(JUMP);
82      reset=1;//边缘触发
83      #(PERIOD*40);
```

Core of the testbench is shown here. It can be concluded into 3 periods.

1) Data=10100101(a5)->11110000(f0) ->00001111(0f) interrupt-mode
2) Data=00001111(0f) ->11000011(c3) polling-mode
3) Data=11000011(c3) interrupt mode

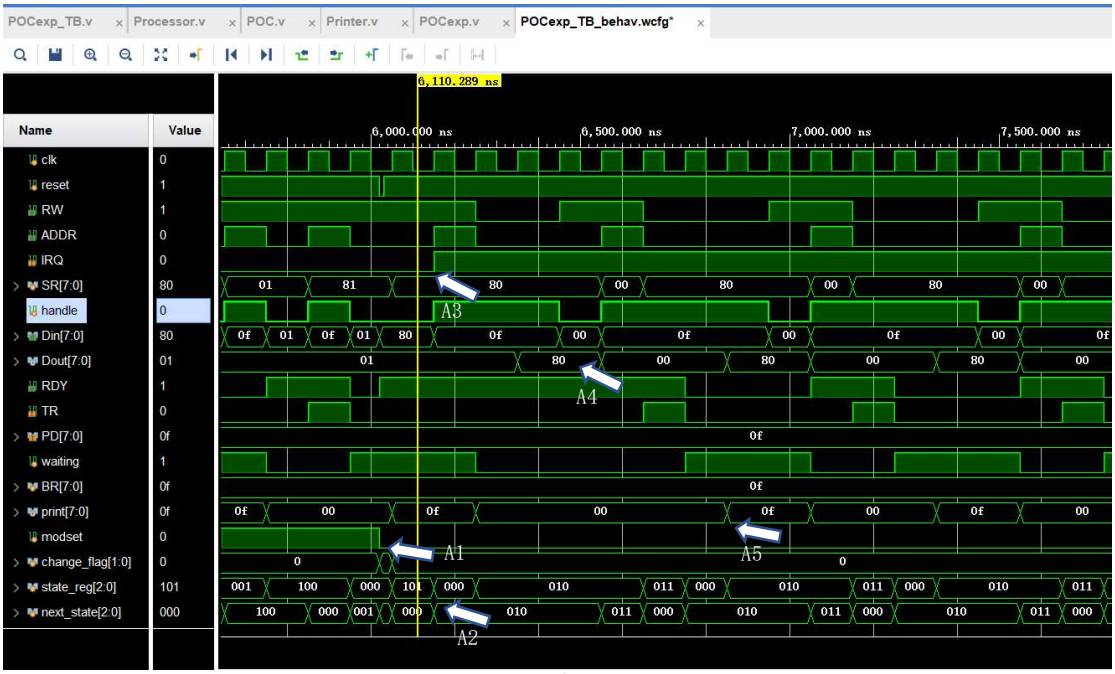
The whole simulation waveform is shown below.

4.2 Simulation Results



We can clearly figure out that it experienced 3 period of different modes by observing IRQ. We can also clearly figure out that the transmitted data has four different values by observing PD.

Let's further discuss the waveforms by focusing on the two moments when two modes alters each other.



1) The first time of mode changes

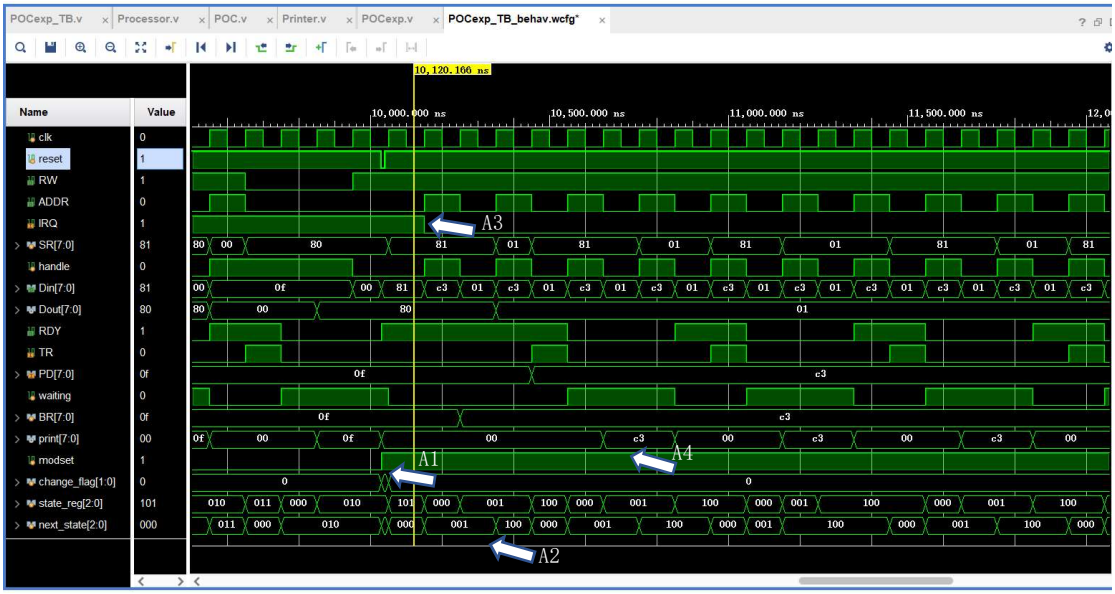
A1: At this special moment, we can see that modset changes from 1 to 0, which means from interrupt mode to polling mode.

A2: Here we notice that after reset, the fsm in POC restart from state6, it reads SR set by CPU, and jumps to state3 then state4.

A3: First RW=1, ADDR=0, CPU writes 10000000 through port Din to SR. Then SR=80'h and IRQ turns to 1.

A4: In polling mode, every time CPU writes something to POC, it first checks if POC is ready through port Dout. Hence we can see a loop of 80 and 00 in the value of Dout

A5: The printer is designed to give 00000000 when not printing, when printing, it costs 3 clocks as we can see here.



2) The second time of mode changes

A1: At this special moment, we can see that modset changes from 0 to 1, which means from polling mode to interrupt mode.

A2: Here we notice that after reset, the fsm in POC restart from state6, it reads SR set by CPU, and jumps to state2 then state5.

A3: First RW=1, ADDR=0, CPU writes 10000001 through port Din to SR. Then SR=81'h and IRQ turns to 0.

A4: At this moment, the author intentionally changes input data and mod at the same time to test the robustness of the design. We can find out that the output data also changed from 0f'h to c3'h after reset signal.

5. Conclusion

1. From the this Design, I've recognized that POC is of great significant in the communication between CPU and I/O. It not only plays a role as a buffer, but adjusts the data exchanging rate as well.

2. A matched time flow, which leads to better synchronousness for every parts of a computer system is unavoidable. Proper time sequencing skills help accelerates the rate of I/O stream.

3.In computer system, an important concept is that each time when an operation is done, a signal should be emitted to inform interfaces connected to it. Only then a machine can indeed work as a whole.

6.Discussion

1. Compared to the vast modern computer design strategies, this POC design not a big deal to be discussed. We also have many other I/O mechanisms like DMA and so on. But it doesn't mean this project is meaningless to us. By building the whole system on my own, I've improved my Verilog skills and objectified my recognition to Computer architecture.

2. When coding, I made several mistakes. For example, there was no schematic after synthesis. And I finally found out that this is because I hadn't set an output at TOP level.

Another important concept I've learned from this project is the difference of Finite State Machines, commonly we adopt the 3 segments FSM, but here I adjusted the architecture to fit my design.

3. If there is flaw in this design, I guess the biggest one is its efficiency.

Actually I intentionally made the POC and the Processor two pipelines. Their states change with clock, but there are sometimes that the change of states maybe paralleled but I limited it to linear to keep a stable system.

4. Thanks for reading

6.Appendix

```
module POCexp(
    input clk,
    input reset,
    input modset,
    input [7:0]data,
    output [7:0]print
    );

wire ADDR;
wire TR;
wire RW;
wire IRQ;
wire RDY;
wire [7:0]Din;
wire [7:0]Dout;
wire [7:0]PD;

Processor u_Processor(
    .ADDR(ADDR),//控制SR还是BR，1是BR
    .Dout(Dout[7:0]),
    .Din(Din[7:0]),
    .RW(RW),//控制读写，1是cpu向外写
    .clk(clk),
    .IRQ(IRQ),//IRQ为0代表中断请求
    .reset(reset),
    .modset(modset),
    .data(data[7:0])
    );

POC u_POC(
    .clk(clk),
    .IRQ(IRQ),
    .RW(RW),
    .Din(Din[7:0]),
    .Dout(Dout[7:0]),
    .ADDR(ADDR),
    .RDY(RDY),
    .TR(TR),
    .PD(PD[7:0]),
    .reset(reset)
    );

Printer u_Printer(
    .clk(clk),
    .TR(TR),
    .PD(PD[7:0]),
    .RDY(RDY),
    .PRINT(print[7:0]),
    .reset(reset)

    );
endmodule


module Processor(
    output ADDR,//控制SR还是BR，1是BR
    input [7:0]Dout,
    output [7:0]Din,
    output RW,//控制读写，1是cpu向外写
    input clk,
    input IRQ,//IRQ为0代表中断请求
    input reset,
    input modset,//1中断0查询
    input [7:0]data
```

```
    );

reg ADDR_reg;
reg RW_reg;
reg [7:0]Din_reg;
reg handle=0;//用于判别cpu是否写入br
//reg check=0;//判别查询模式是否查过
//reg [7:0]data=8'b10100101;
reg [1:0]change_flag=2'b00;//模式切换时空过1slot

always @(negedge reset or posedge clk)
begin
    if(!reset)
    begin
        change_flag<=2'b01;
        if(modset)//如果中断开局
        begin
            ADDR_reg<=0;//写sr
            RW_reg<=1;
            Din_reg<=8'b10000001;
            handle<=0;
        end
        else
        begin//查询模式开局
            ADDR_reg<=0;//写sr
            RW_reg<=1;
            Din_reg<=8'b10000000;
            handle<=0;
        end
    end
    else
//      begin

//      end
//end
//          //reset根据mod重置为默认
//always @(posedge clk)
    begin
        if(change_flag==2'b01)
        begin
        //空置时隙
            change_flag<=2'b00;
        end
        else if(change_flag==2'b00)
        begin
            if(!IRQ)//中断请求
            begin
                if(handle==0)
                begin

ADDR_reg<=1;
                    RW_reg<=1;

Din_reg<=data;//写入data
                    handle<=1;
                end
                else
                begin

ADDR_reg<=0;
                    RW_reg<=1;

Din_reg<=8'b00000001;//SR0=1,SR7=0
;
                    handle<=0;//sr
处理完毕
                end
            end
            /*
            else
if(RW&&Dout==8'b00000001)//没准备
好
            begin
                RW_reg=0;
            end
            */
            else if(IRQ)//查询
            begin

if(Dout==8'b10000000)
                //查询sr7
                begin
                    if(handle==0)
                    begin
                        RW_reg
```

```
<= 1;

ADDR_reg <= 1;

Din_reg<=data;

handle<=1;
                        end
                        else
                        begin//置sr7=0
                            RW_reg
<=1;

ADDR_reg <=0;

Din_reg<=8'b00000000;

handle<=0;
                        end
                    end
                    else
                    begin//保持读取态
查询
                        RW_reg<=0;

ADDR_reg<=0;
                    end
                end
                else
                begin
                    //空过
                end
            end
        end
end

assign ADDR = ADDR_reg;
assign RW = RW_reg;
assign Din = Din_reg;
endmodule

module POC(
    input clk,
    output IRQ,
    input RW,
    input [7:0]Din,
    output [7:0]Dout,
    input ADDR,
    input RDY,
    input reset,
   // input mod,//1中断0查询
    output TR,
    output [7:0]PD
    );
reg IRQ_reg;
reg TR_reg;
reg[7:0] PD_reg;
reg[7:0] Dout_reg;
reg[2:0] state_reg;
reg[2:0] next_state=3'b000;
reg[7:0] BR;
reg[7:0] SR;

parameter state1= 3'b000;
parameter state2= 3'b001;
parameter state3= 3'b010;
parameter state4= 3'b011;
parameter state5= 3'b100;
parameter state6= 3'b101;//重开模式

always@(negedge reset or posedge clk)
begin
   // mod_reg<=mod;//首先载入模式
    if(!reset)
    begin
        next_state=state6;//先读取模
式
    end
   // mod_reg<=0;//默认查询
    else
    begin
        state_reg=next_state;
    end
end
always@(posedge clk)
begin
```

```
        //state_reg=next_state;
        case(state_reg)
            state1:
            begin
                TR_reg<=0;//复原TR脉
冲
                if(SR==8'b10000001)
                begin
                    IRQ_reg<=0;
                    next_state<=state2;
                end
                else
if(SR==8'b10000000)
                begin
                    IRQ_reg<=1;
                    next_state<=state3;
                end
                else
if(SR==8'b00000000)
                begin
                    IRQ_reg<=1;
                    next_state<=state4;
                end
                else
if(SR==8'b00000001)
                begin
                    IRQ_reg<=0;
                    next_state<=state5;
                end
                else//reset模式尝试
                begin
                    next_state<=state6;
                end
            end
            state2://中断
            begin

if(RW==1&&ADDR==1)
                begin
                    BR<=Din;
                    //SR<=8'b00000001;
                   // next_state=state2;
                end
                else
if(RW==1&&ADDR==0)
                begin
                    SR<=Din;
                    Dout_reg<=Din;
                    next_state <=state5;
                end
            end
            state3://查询
            begin

if(RW==0&&ADDR==0)//读sr
                begin
                    Dout_reg<=SR;
                    //next_state=state3;
                    //next_state
<=state3;
                end
                else
if(RW==1&&ADDR==1)//写br
                begin
                    BR<=Din;
                    //next_state=state3;
                    //next_state
<=state3;
                end
                else
if(RW==1&&ADDR==0)//写sr
                begin
                    SR<=Din;
                    Dout_reg<=Din;
                    next_state<=state4;
                end
            end
            state4://查询后握手
            begin
                if(RDY)
                begin
                    TR_reg<=1;
                    PD_reg<=BR;
                    SR<=8'b10000000;
                    next_state<=state1;
```

```
            end
        end
        state5://中断后握手
        begin
            if(RDY)
            begin
                TR_reg<=1;
                PD_reg<=BR;
                SR<=8'b10000001;
                next_state<=state1;
            end
        end
        state6:
        begin
            SR<=Din;
            next_state<=state1;
        end
        default:
            next_state<=state6;
    endcase
end

assign IRQ = IRQ_reg;
assign TR = TR_reg;
assign PD = PD_reg;
assign Dout = Dout_reg;

endmodule


module Printer(
    input clk,
    input TR,
    input [7:0]PD,
    input reset,
    output RDY,
    output [7:0]PRINT
    );
reg [7:0]data;//data是转存
reg [7:0]print;//print是等待后打印输出
reg [2:0]counter=3'b000;
//计时器
reg waiting=0;
//写入过程的waiting，写入时为1
reg ready;
//RDY的赋值版本

always @(posedge clk or negedge reset)
begin
    if(!reset)
    begin
        print<=0;
        ready<=1;
    end
    else
    begin
        if(TR)
        begin
            ready<=0;
            data<=PD;
            waiting<=1;
            //如果TR已经触发，不
再接收，开始等待延时
        end
        else//TR没触发
        begin
            if(waiting)//如果是等待
中
            begin

counter<=counter+1;
                print<=data;
                if(counter==3'b010)
                begin
                    print<=0;
                    ready=1;
                    waiting<=0;
                    counter<=0;
                end
            end
        end
    end
end

assign RDY = ready;
assign PRINT = print;
endmodule
```