嵌入式第二次作业

林逸晗 2016010766

1、自己下载74LS165芯片资料，根据资料利用Verilog代码编写一个8位并转串芯片74LS165，实现该芯片全部功能。

搜索到的部分资料如下：

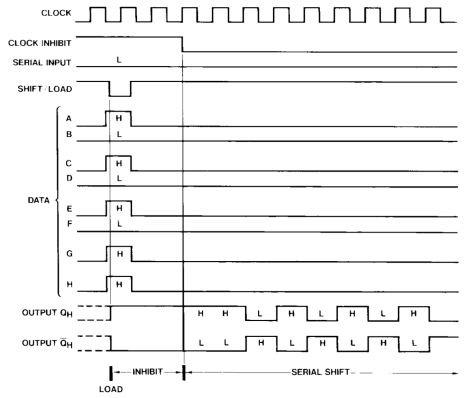
功能表以及工作时序图

Function Table

Inputs					Internal Outputs		Output
Shift/ Load	Clock Inhibit	Clock	Serial	Parallel A...H	Q_A	Q_B	Q_H
L	X	X	X	a...h	a	b	h
H	L	L	X	X	Q_{A0}	Q_{B0}	Q_{H0}
H	L	↑	H	X	H	Q_{An}	Q_{Gn}
H	L	↑	L	X	L	Q_{An}	Q_{Gn}
H	H	X	X	X	Q_{A0}	Q_{B0}	Q_{H0}

H = HIGH Level (steady state)
L = LOW Level (steady state)
X = Don't Care (any input, including transitions)
↑ = Transition from LOW-to-HIGH level
a...h = The level of steady-state input at inputs A through H, respectively.
Q_{A0}, Q_{B0}, Q_{H0} = The level of Q_A, Q_B, or Q_H, respectively, before the indicated steady-state input conditions were established.
Q_{An}, Q_{Gn} = The level of Q_A or Q_G, respectively, before the most recent ↑ transition of the clock.

Timing Diagram



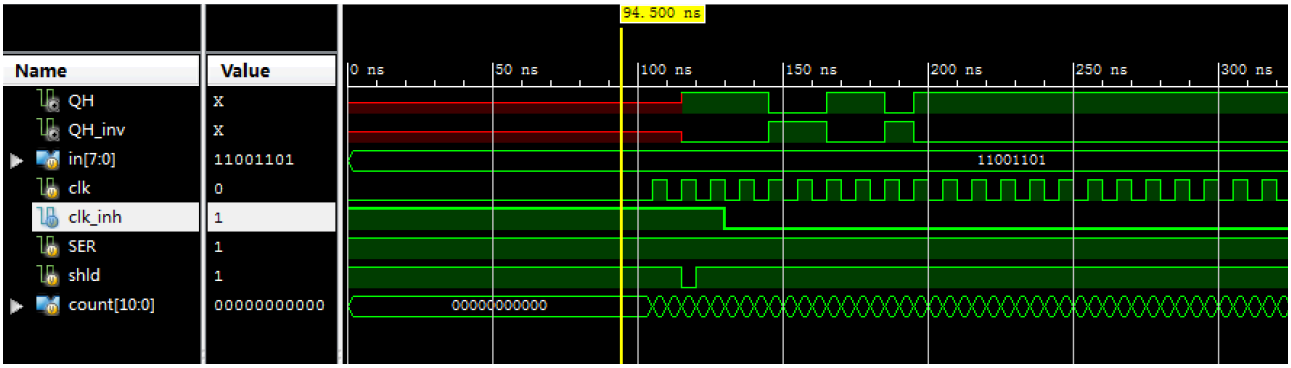
Typical Shift, Load, and Inhibit Sequences

代码如下：（homework1/my74ls165.v）

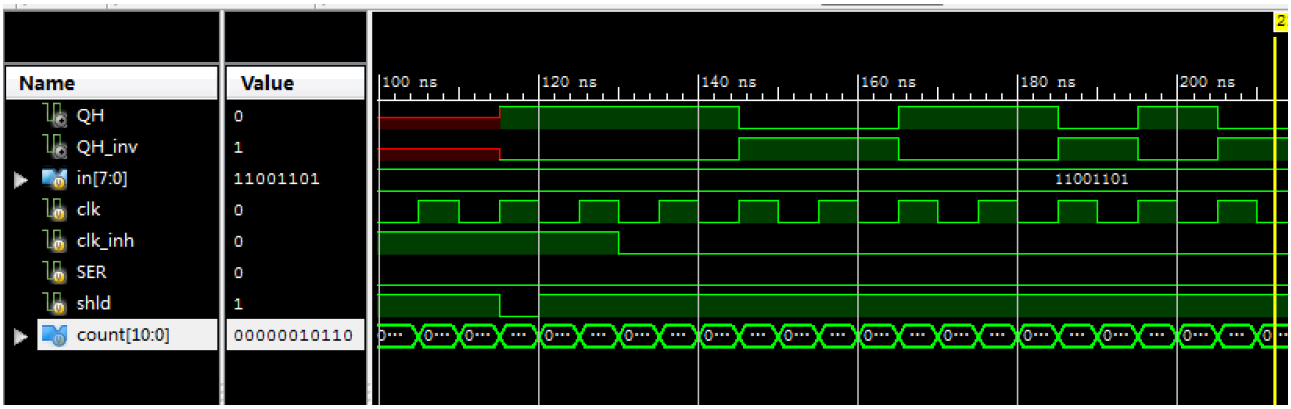
```
module my74ls165(in,clk,clk_inh,SER,shld,QH,QH_inv);
input[7:0] in;
input clk;
input clk_inh;
input SER;
input shld;
output QH;
output QH_inv;
reg[7:0] regs;
reg[3:0] count;
assign QH = regs[7];
assign QH_inv = ~QH;
always@(posedge clk)
begin
        if(~shld)
        begin
                regs <= in;      //Dtrigger, work without clk
        end
        else if(!clk_inh)          //inh, prepare the data
        begin
                regs <= (regs << 1);
                regs[0] <= SER;          //serial  级联进位
        end
end

endmodule
```

用于仿真的代码见附件tb_74ls165_testbench.v，使用clk实现了datasheet中的时序输入，有如下输出：

1_input=11001101,SER=1



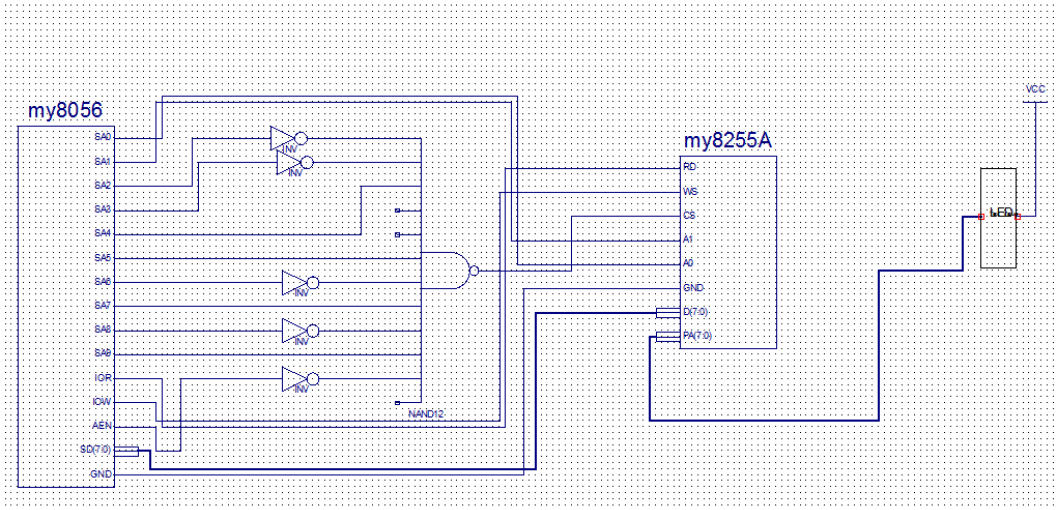
1_input=11001101,SER=0



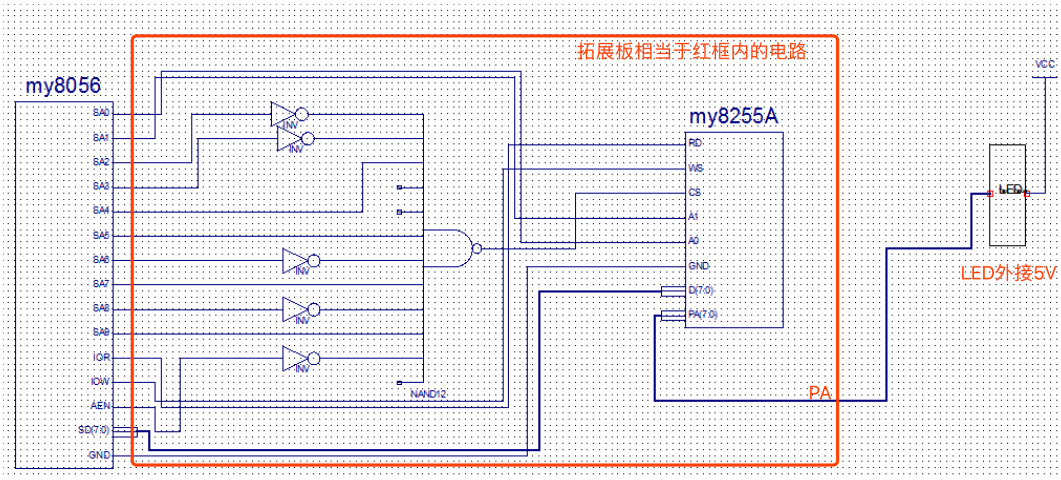
和datasheet中所述功能表一致，完成了并转串的时序，取反输出以及级联模式下的工作。

2、设计一块PC104总线版本1下的扩展板卡原理图，该板卡利用8255A工作方式0控制8个LED灯的开关。要求该板卡的基址为0x2B0。注意：不用Verilog代码设计，要用第一讲中约定的各种门电路符号设计接口电路原理图。

使用ISE布置电路图如下：



说明如下：



板卡部分代码为bk.v：

```
module bk(
    input [9:0] SA,
    input [7:0] SD,
    input IOR,
    input IOW,
    input AEN,
    input GND,
    output [7:0] PA
    );
    wire CS;
    wire andAS;
    wire a3,a2,a6,a8,AEN_inv;

    assign a3 = ~SA[3];
    assign a2 = ~SA[2];
    assign a6 = ~SA[6];
    assign a8 = ~SA[8];
    assign AEN_inv = ~AEN;
    nand(andAS,a3,a2,a6,a8,SA[4],SA[5],SA[7],SA[9],AEN_inv);
    assign CS <= andAS;
    if(~IOR && ~SA[0] && ~SA[1] && ~CS)
    begin
        assign PA <=SD;
    end
endmodule
```

3、利用Verilog代码实现一个类似于8255A工作方式1的用户自定义并口芯片，其中该芯片的1个8位PA口工作在方式1的输入模式，1个8位PB口工作在方式1下的输出模式。要求该芯片可以直接与PC104总线连接，芯片基址为0x2B0。

方式一使用C口作为控制端口。其中：
/RD，STB，IBF，INTRA作为PA的控制端口。
/WR，ACK，OBF，INTRB作为PB的控制端口。

所需完成的任务有：

1、片选信号生成
2、PA，PB，StateWord寄存器的选择。
3、PA功能时序实现
4、PB功能时序实现

所写代码如下（附件homework3/my8255A.v）注释中解释代码功能

```
module my8255A(
    input [9:0] SA,
    inout [7:0] SD,
          input [7:0] PA,
          output INTR_A,
          input  INTE_A,
          input  STB,
          output IBF_A,
          output [7:0] PB,
          output INTR_B,
          input INTE_B,
          output OBF_B,
          input ACK,
          input RST,
    input IOR,
    input IOW,
    input AEN,
    input GND
    );  //定义各个引脚

        wire CS,CSA,CSB,CSW;      //片选信号线
        wire a0,a3,a2,a6,a8,a1;   //为了增强可读性引入
        reg [7:0] stateWord;      //状态字，选择工作方式1
        reg [7:0] PB_reg;         //PB所用的缓冲寄存器
        reg [7:0] PA_reg;         //PA所用的缓冲寄存器
        //定义各个引脚的类型
        wire [7:0] PA;
        wire [7:0] PB;

        reg INTR_A;  //to_PC
        reg IBF_A;
        wire STB;
        wire INTE_A;
        reg INTR_B; //to_peri
```

```
        wire INTE_B;
        wire ACK;
        reg OBF_B;

//译码器部分
        assign a0 = ~SA[0];
        assign a1 = ~SA[1];
        assign a2 = ~SA[2];
        assign a3 = ~SA[3];
        assign a6 = ~SA[6];
        assign a8 = ~SA[8];
        assign AEN_inv = ~AEN;
        assign CS_inv = ~CS;
        nand(CS,SA[9],a8,SA[7],a6,SA[5],SA[4],a3,a2,AEN_inv);
        and(CSA,CS_inv,a0,a1);                  //由1010110000生成CSA信号，选择PA口
        and(CSB,CS_inv,a0,SA[1]);               //由1010110010生成CSB信号，选择PB口
        and(CSW,CS_inv,SA[0],SA[1]);            //由1010110011生成CSW信号，选择状态字。


//PA,PB的三态阻塞
        bufif0(PB[0],PB_reg[0],stateWord[1]); //if stateWord[1]==0, output
        bufif0(PB[1],PB_reg[1],stateWord[1]); //if stateWord[1]==0, output
        bufif0(PB[2],PB_reg[2],stateWord[1]); //if stateWord[1]==0, output
        bufif0(PB[3],PB_reg[3],stateWord[1]); //if stateWord[1]==0, output
        bufif0(PB[4],PB_reg[4],stateWord[1]); //if stateWord[1]==0, output
        bufif0(PB[5],PB_reg[5],stateWord[1]); //if stateWord[1]==0, output
        bufif0(PB[6],PB_reg[6],stateWord[1]); //if stateWord[1]==0, output
        bufif0(PB[7],PB_reg[7],stateWord[1]); //if stateWord[1]==0, output
        bufif1(SD[0],PA_reg[0],stateWord[4]); //if stateWord[4]==1, output
        bufif1(SD[1],PA_reg[1],stateWord[4]); //if stateWord[4]==1, output
        bufif1(SD[2],PA_reg[2],stateWord[4]); //if stateWord[4]==1, output
        bufif1(SD[3],PA_reg[3],stateWord[4]); //if stateWord[4]==1, output
        bufif1(SD[4],PA_reg[4],stateWord[4]); //if stateWord[4]==1, output
        bufif1(SD[5],PA_reg[5],stateWord[4]); //if stateWord[4]==1, output
        bufif1(SD[6],PA_reg[6],stateWord[4]); //if stateWord[4]==1, output
        bufif1(SD[7],PA_reg[7],stateWord[4]); //if stateWord[4]==1, output

// defaut: choose mode_1
        always@(CSW)
        begin
                stateWord = 8'b11111111;
                stateWord[4] =1;
                stateWord[1] =0;
        end

//PA的时序处理部分，使用边沿触发完成时序编写，将同一寄存器的赋值合并后通过if语句分支实现
        always@(negedge STB or posedge IOR or posedge RST)
        begin
                if (RST)
                begin
                        PA_reg <= 8'b00000000;   //peri send data
                        IBF_A<=0;
                        $display("RSTA");
                        //RST复位信号
                end
                else if (~STB)
                begin
```

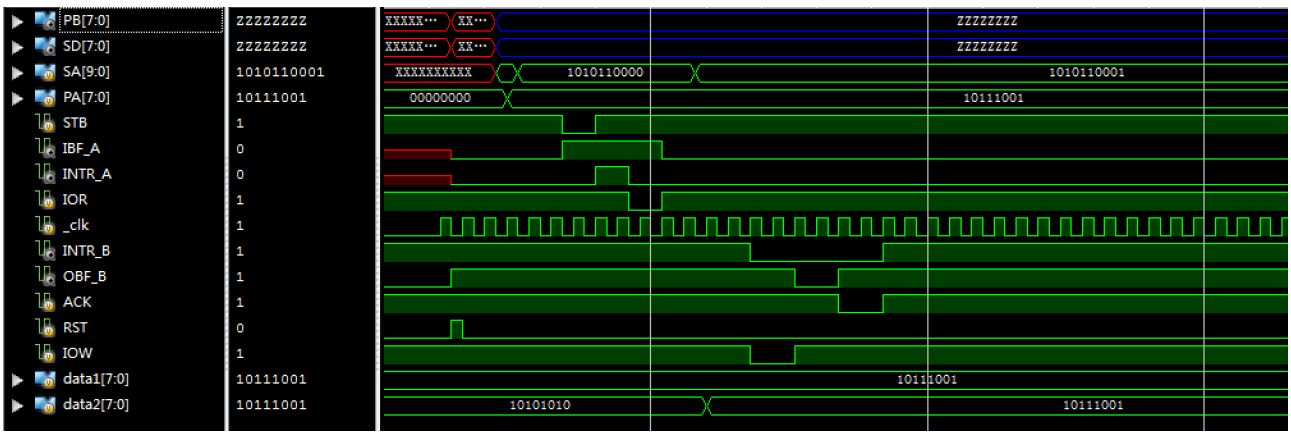
```
                PA_reg <= PA;   //peri send data
                IBF_A<=1;               //buffered
                $display("step1,store data",IBF_A);
                //第1步，存储信号，缓存满
        end
        else if (IOR)
        begin
                if(CSA)
                        IBF_A<=0;               //buffered
                $display("step 4,INF= ",IBF_A);
                //第4步，缓存空，可以补充清空reg语句，但没必要

        end
end

always@(posedge STB or negedge IOR or posedge RST)
begin
        if (RST)
        begin
                $display("RSTA");
                INTR_A<=0;                      //复位
        end else if (~IOR)
        begin
                if(CSA)
                        INTR_A<=0;              //buffered
                $display("step 3,INTR= ",INTR_A);
                //第3步，终止中断

        end
        else if (STB)
        begin
                if(IBF_A && INTE_A && IOR)
                        INTR_A<=1;              //buffered
                $display("step2 en-inter ",INTR_A);
                $display("reg=",PA_reg,"  SD=",SD);
                //第2步，开始中断，调试查看reg的信息
        end
end
```

//PB的时序处理部分，使用边沿触发完成时序编写，将同一寄存器的赋值合并后通过if语句分支实现

```
always@(negedge IOW or posedge ACK or posedge RST)
begin
if (RST)
        begin
                $display("RSTB");
                //复位
                INTR_B<=1;
                PB_reg<=0;
        end
else if (~IOW)
        begin
                $display("step1= ", PB_reg);
                //第1步，复位中断，存储数据
                INTR_B <=0;             //start task of peri
                PB_reg <= SD;           //CPU data sent to port B tri-regs
        end
else
```

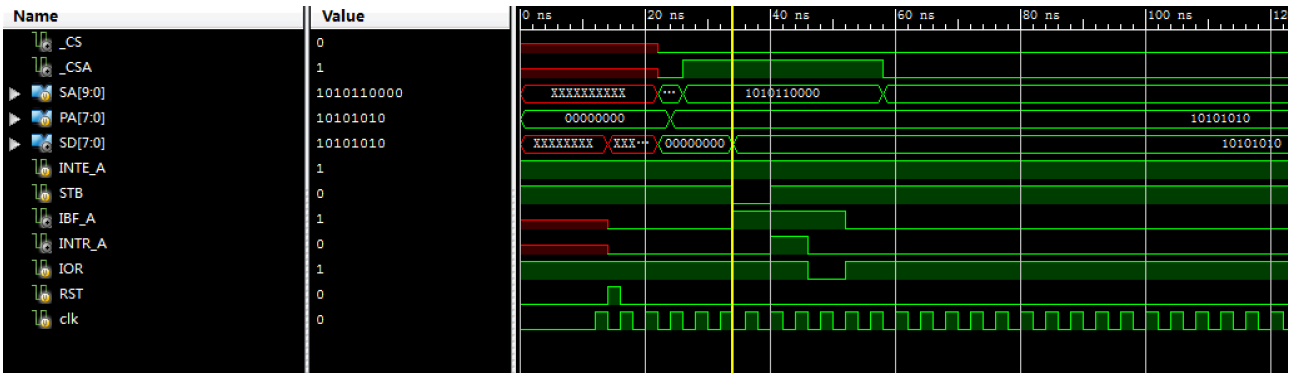
```
        begin
                INTR_B <=1;
                $display("step3");
                //第3步，复位中断
        end
end

always@(posedge IOW or negedge ACK or posedge RST)
begin
if (RST)
begin
        $display("RSTB");
        //复位
        OBF_B <=1;
end
else if (ACK)
        begin
                OBF_B <=0;
                //第2步，缓存空
                $display("step2 reg=", PB_reg);
        end
else if(~ACK)
        //第4步，复位等待
        begin
                OBF_B <=1;
                $display("step4 ");
        end
end


endmodule
```

所得时序图如下：（仿真文件为homework3/new_test.v）

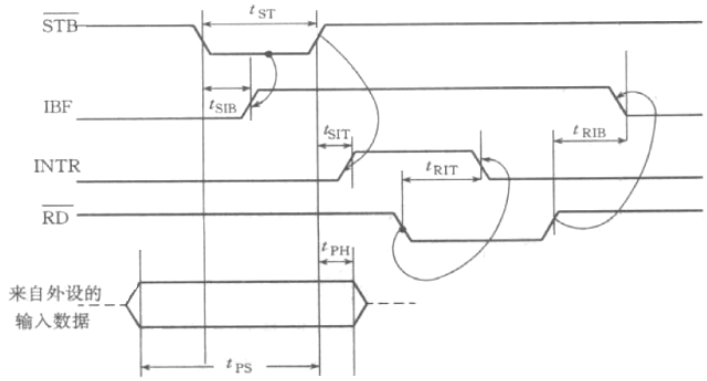


仿真的流程：
复位-> 写状态字 -> PA口输入data1-> PB口输出

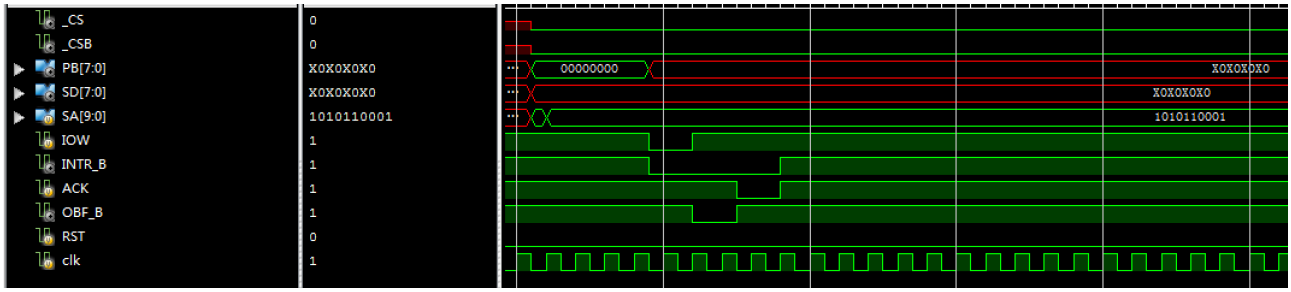
PA口时序如图



可以看到，诸位寄存器的工作时序和原厂功能时序匹配，并且在触发后也完成了PA口数据并行传输到SD口的功能。对比：



PB口时序如图：



对比：