

**本科实验报告**

**课程名称： 数字逻辑电路设计**

**姓 名：**

**学 院： 计算机科学与技术学院**

**专 业： 计算机科学与技术**

**邮 箱：**

**QQ 号：**

**电 话：**

**指导教师：**

**报告日期： 2022 年 12 月 11 日**

**浙江大学实验报告**

课程名称： 数字逻辑设计 实验类型： 综合

实验项目名称： 寄存器和寄存器传输设计

学生姓名： 学号： 同组学生姓名：

实验地点： 线上实验 实验日期： 2022 年 12 月 11 日

# **实验目的和要求**

1. 掌握寄存器传输电路的工作原理
2. 掌握寄存器传输电路的设计方法
3. 掌握ALU和寄存器传输电路的综合应用

# **实验内容和原理**

**2.1 实验任务**

基于ALU的数据传输应用设计

**2.2 实验原理**

2.2.1 寄存器

寄存器是一组二进制存储单元；一个寄存器可以用于存储一列二进制值，通常用于进行简单数据存储、移动和处理等操作。

寄存器可以存储信息并保存多个时钟周期，能用信号来控制“保存”或“加载”信息。

对于采用门控市时钟的寄存器，如果 Load 信号为 1，则允许时钟信号通过，如果 Load 信号为 0 则阻止时钟信号通过。

例如： 对于上升沿触发的边沿触发器或负向脉冲触发的主从触发器示意图如下：

Clock

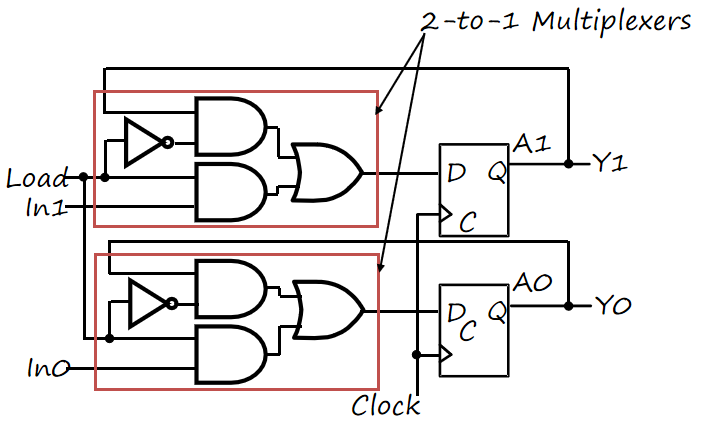
Load

触发器

Gated Clock

Gated Clock = Clock + Load

对于采用Load控制反馈的寄存器，可以进行有选择地加载寄存器且更可靠方法，在保证时钟的连续性的同时，选择性地使用加载控制来改变寄存器的内容，其逻辑电路设计示意图如下：



使用 Verilog 语言描述，定义为 myregister4b module ，具体代码如下：

module myregister4b(

input wire clk,

input wire [3:0] IN,

input wire load,

output reg [3:0] OUT

);

always@(posedge clk)begin

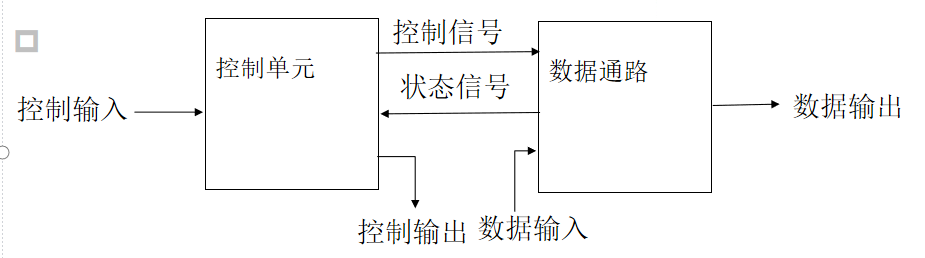
if(load)OUT<=IN;

end

endmodule

2.2.2 寄存器传输

寄存器传输是指寄存器中数据的传输和处理，其具体示意图如下：



对映的三个基本单元分别是：寄存器组、操作、操作控制。

对映的基本操作为:加载、计数、移位、加法、按位操作等。

据此制造采用寄存器传输原理的计数器，其运行架构示意图如下：



其中，SW[2] 上下拨动一次，表示计一次数（将当前自增/减值传入寄存器内），Load 控制模块表示当 SW[2] 打开时，若遇到 clk 时钟上升沿则产生1个时钟周期宽度的 Load 信号，使寄存器中数值更新；另外，自增/自减计数器可以用 4 位加减法器实现。

SW[0] 关于自增/减的调控，当 SW[0] 打开时，表示作自减操作；当 SW[1] 关闭时，表示作自增操作。

SW[15] 控制寄存器的更新数值取向状态，若 SW[15] 处于打开状态，则当Load 信号来临时，更新自增/减数值；若 SW[15] 闭合，则将寄存器更新为0.

首先需要设计具体的 Load信号产生条件，其对映的 Verilog 代码设计如下：

module load\_gen(

input wire clk,

input wire btn\_in,

output reg load\_out

);

initial load\_out=0;

wire btn\_out;

reg old\_btn;

assign btn\_out=btn\_in;

always@(posedge clk)begin

if((old\_btn==1'b0)&&(btn\_out==1'b1))

load\_out<=1'b1;

else

load\_out<=1'b0;

end

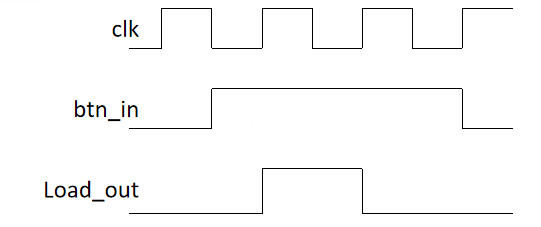
always@(posedge clk)begin

old\_btn<=btn\_out;

end

endmodule

上述代码对映的时钟图像表达见下：



在获得一定条件下产生的寄存器写入信号 Load 后，我们可以采取一下的Verilog 代码进行基于 SW[15] 开合状态的 A 寄存器赋值：

myregister4b m2(.clk(clk),.IN(Ain),.load(loadA),.OUT(A));

load\_gen m3(.clk(clk),.btn\_in(SW[2]),.load\_out(loadA));

bit4addsub m4(.A(A),.B(4'b0001),.fu(SW[0]),.S(A1),.Cout(c0));

assign Ain=(SW[15]==1'b0)?A1:4'b0000;

其中，bit4addsub module 为前述实验中设计模块，其功能为对两个给定的 4 位操作数进行加或减的操作；其中 register4b 为实验原理 2.2.1 中设计模块。

2.2.3 基于多路选择器总线的寄存器传输

由一个多路选择器驱动的总线不仅可以同功能选择阐述的数据，还可以降低硬件开销；但是这样的结构不能实现多个寄存器相互之间的并行传输操作，其具体设计示意图如下：

L0

n

n

MUX

S1 S0

0

1

n

2

S0

S1

L1

L2

n

R0

Load

n

R1

Load

n

R2

Load

n

总体的逻辑关系设计图如下：



在上述逻辑关系中，SW[2]、 SW[3] 和 SW[4] 分别控制寄存器 A 、寄存器 B 和寄存器 C 的 Load 信号的产生，SW[15] 控制寄存器 A 和 B 的存入数据选择，当 SW[15] 为 0 时，选用自增/自减数据，当 SW[15] 为 1 时，将寄存器置为主线上的数值；SW[0],SW[1]分别控制寄存器 A 和寄存器 B 的自增或自减效果。

3 选 1 多路复用器的部分设计可以使用已有的 4 选 1 模块 Mux4to14b，在选择时将 i3 默认置为 4’b1111 通过 SW[8:7] 作为选择信号，选择需要的寄存器对映数值。

具体的 Verilog 代码实现如下：

module translate(

input clk,

input [15:0] SW,

output [3:0] AN,

output [7:0] SEGMENT);

wire loada,loadb;

wire [3:0]A,A1,Ain;

wire [3:0]B,B1,Bin;

wire [3:0]C,Cin;

wire ca,cb;

wire [3:0] num;

Mux4to14b m1(.i0(A),.i1(B),.i2(C),.i3(4'b1111),

.s(SW[8:7]),.o(num[3:0]));

myregister4b a1(.clk(clk),.IN(Ain),.load(loada),.OUT(A));

load\_gen a2(.clk(clk),.btn\_in(SW[2]),.load\_out(loada));

bit4addsub a3(.A(A),.B(4'b0001),.fu(SW[0]),.S(A1),.Cout(ca));

assign Ain=(SW[15]==1'b0)?A1:num[3:0];

myregister4b b1(.clk(clk),.IN(Bin),.load(loadb),.OUT(B));

load\_gen b2(.clk(clk),.btn\_in(SW[3]),.load\_out(loadb));

bit4addsub b3(.A(B),.B(4'b0001),.fu(SW[1]),.S(B1),.Cout(cb));

assign Bin=(SW[15]==1'b0)?B1:num[3:0];

myregister4b c1(.clk(clk),.IN(Cin),.load(loadc),.OUT(C));

load\_gen c2(.clk(clk),.btn\_in(SW[4]),.load\_out(loadc));

assign Cin=(SW[15]==1'b0)?4'b0000:num[3:0];

disp\_num m2(clk,{A,B,C,4'b0000},4'b0000,4'b0000,1'b0,AN,SEGMENT);

endmodule

其中，Mux4to14b 模块为课程前述设计实验中 4 选 1 多路复用器模块，功能为根据选择信号，4 选 1 选择指定数值；load\_gen 模块，bit4addsub 模块和 myregister4b 模块在本实验报告前述部分均已给出；disp\_num 模块同样为课程前述设计实验中四位七段数码管显示模块，其功能为将指定的 16 位数字按 4 位分割逐位输出至七段数码管上。

2.2.4 寄存器传输应用设计

结合第二与第三部分功能，其逻辑关系设计示意图如下：



在上述逻辑关系中，SW[2]、 SW[3] 和 SW[4] 同样分别控制寄存器 A 、寄存器 B 和寄存器 C 的 Load 信号的产生，SW[15] 控制寄存器 A 、寄存器 B和寄存器 C 的存入数据选择，当 SW[15] 为 0 时，对于寄存器 A,B 表示选用自增/自减数据，对于寄存器 C 表示选择寄存器 A 与 B 中数值经过 ALU 模块相应操作后数值；当 SW[15] 为 1 时，表示将寄存器 A, B, C 中数值置为主线上的数值；SW[0],SW[1]分别控制寄存器 A 和寄存器 B 的自增或自减效果。

3 选 1 多路复用器的部分设计可以使用已有的 4 选 1 模块 Mux4to14b，在选择时将 i3 默认置为 4’b1111 通过 SW[8:7] 作为选择信号，选择需要的寄存器对映数值，将其输出至主线 BUS 上；使用 SW[6:5] 的开合输出 ALU 运算的选择信号，00--加法，01--减法，10--或操作，11--和操作。

具体的 top module 代码实现如下：

module final(

input wire clk,

input wire [15:0] SW,

output wire [15:0] total);

wire loada,loadb;

wire [3:0]A,A1,Ain;

wire [3:0]B,B1,Bin;

wire [3:0]C,Cin;

wire ca,cb,ct;

wire [7:0] num;

Mux4to14b m1(.i0(A),.i1(B),.i2(C),.i3(4'b0000),

.s(SW[8:7]),.o(num[3:0]));

ALU m2(.A(A),.B(B),.S(SW[6:5]),.C(num[7:4]),.C0(ct));

myregister4b a1(.clk(clk),.IN(Ain),.load(loada),.OUT(A));

load\_gen a2(.clk(clk),.btn\_in(SW[2]),.load\_out(loada));

bit4addsub a3(.A(A),.B(4'b0001),.fu(SW[0]),.S(A1),.Cout(ca));

assign Ain=(SW[15]==1'b0)?A1:num[3:0];

myregister4b b1(.clk(clk),.IN(Bin),.load(loadb),.OUT(B));

load\_gen b2(.clk(clk),.btn\_in(SW[3]),.load\_out(loadb));

bit4addsub b3(.A(B),.B(4'b0001),.fu(SW[1]),.S(B1),.Cout(cb));

assign Bin=(SW[15]==1'b0)?B1:num[3:0];

myregister4b c1(.clk(clk),.IN(Cin),.load(loadc),.OUT(C));

load\_gen c2(.clk(clk),.btn\_in(SW[4]),.load\_out(loadc));

assign Cin=(SW[15]==1'b0)?num[7:4]:num[3:0];

assign total={A,B,C,num[3:0]};

endmodule

大部分模块已在 2.2.3 的讲解中注明，其中 ALU 模块同为课程前述设计模块，其功能为根据输入的选择信号，对两个输入操作数进行对映的逻辑操作，00--加法，01--减法，10--或操作，11--与操作，并将结果与对应的进位信息输出。

1. **主要仪器设备**
2. 装有 Xilinx ISE 14.7 的计算机 1台
3. SWORD 开发板 1套
4. **操作方法与实验步骤**

4.1 采用寄存器传输原理设计计数器

1.验证寄存器的设置初值功能

2.验证寄存器自增、自减功能

其对映的 top module 设计为：

module addnew(

input wire clk,

input wire [15:0]SW,

output wire [15:0] num);

wire loadA,c0;

wire[3:0] A,Ain,A1;

wire [31:0] clkdiv;

clkdivna m1(clk,1'b0,clkdiv);

myregister4b m2(.clk(clk),.IN(Ain),.load(loadA),.OUT(A));

load\_gen m3(.clk(clk),.btn\_in(SW[2]),.load\_out(loadA));

bit4addsub m4(.A(A),.B(4'b0001),.fu(SW[0]),.S(A1),.Cout(c0));

assign Ain=(SW[15]==1'b0)?A1:4'b0000;

assign num={A,A1,Ain,4'b0000};

endmodule

对此模块进行仿真，仿真的具体代码如下：

initial forever begin

clk=0;#5;

clk=1;#5;

end

initial begin

SW=0;

SW[15]=1;

SW[2]=0;#20;

SW[2]=1;#10;

SW[2]=0;#20;

SW[15]=0;

SW[0]=0;

SW[2]=1;#10;SW[2]=0;#20;

SW[2]=1;#10;SW[2]=0;#20;

#50;

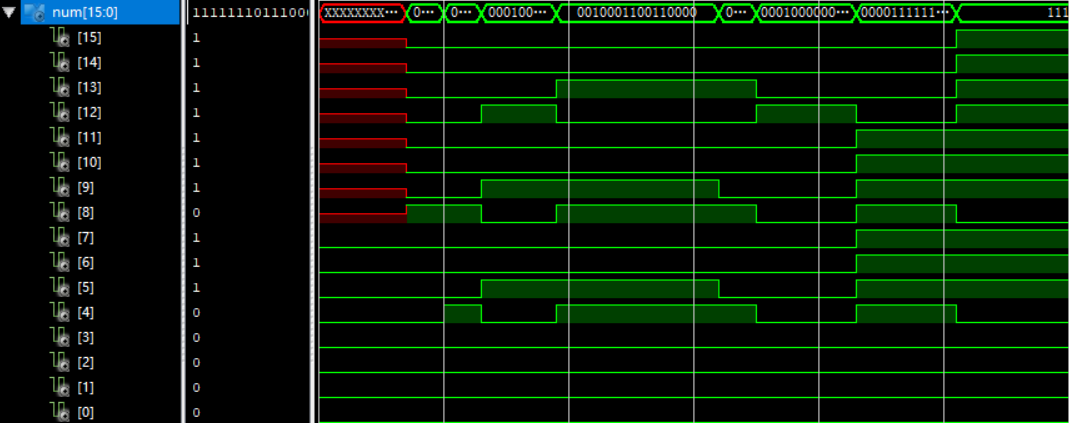
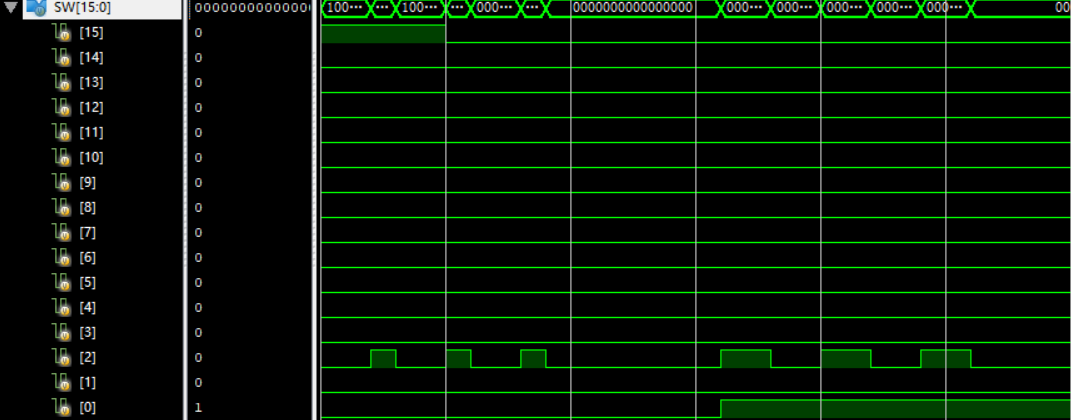
SW[0]=1;SW[2]=1;#20;SW[2]=0;#20;

SW[2]=1;#20;SW[2]=0;#20;

SW[2]=1;#20;SW[2]=0;#20;

end

其仿真结果如下：



在本模块实验中，因为是通过仿真验证程序代码的正确性，因而不能使用disp\_num 模块，需要设定一个额外的输出值，通过 assign 语句实时接收寄存器或中间变量的变化，其中 num[15:12] 表示寄存器A中数值，num[11:8] 表示寄存器对映自增自减保留数的对映数值，num[7:4] 表示，num[3:0] 恒为0.

在本次具体的仿真中，并没有自动将寄存器处于 4’b0000 的初始状态，而是采用手动清零的方法；从仿真的波形图中可以见到，通过将 SW[15] 置 1，再给予 SW[2] 上升沿信号，则可讲寄存器中数值置 0；接着，将 SW[15] 置 1，使 SW[0] 处于闭合状态，表示 bit4addsub 模块作自增，随着 SW[2] 每上下拨动一次，寄存器 A 中数值作自增加 1，寄存器 A 对映的自增操作保留数也发生相应的变化；此时将 SW[0] 打开，对映于自减操作，可以发现此时自增自减保留数已发生相应的变化，随着 SW[2] 的上下拨动，寄存器中数值不断发生减 1 操作，数值输出符合仿真实验要求与预期，说明程序代码设计正确。

4.2 基于多路选择器总线的寄存器传输

1.验证A、B、C寄存器的设置初值功能

2.验证A、B寄存器的自增、自减功能

3.验证A、B、C寄存器之间的传输功能

本实验具体代码实现已在 2.2.3 部分中给出并加以阐述，且不作仿真实验的要求。

4.3 基于ALU的数据传输应用设计

1.验证A、B、C寄存器的设置初值功能

2.验证A、B寄存器的自增、自减功能

3.验证ALU运算功能

4.验证寄存器传输功能

具体的 Verilog 代码已在上述 2.2.4 部分中给出，下面给出本实验对映的仿真代码：

initial forever begin

clk=0;#5;

clk=1;#5;

end

initial begin

SW=0;

SW[15]=0;

#50;

SW[2]=1;#10;

SW[2]=0;#10;

SW[2]=1;#10;

SW[2]=0;#10;

SW[2]=1;#10;

SW[2]=0;#10;

SW[1]=1;

SW[3]=1;#10;

SW[3]=0;#10;

SW[3]=1;#10;

SW[3]=0;#10;

SW[6:5]=2'b01;

SW[4]=1;#10;

SW[4]=0;#10;

SW[8:7]=2'b10;

SW[15]=1;

SW[2]=1;#10;

SW[2]=0;#10;

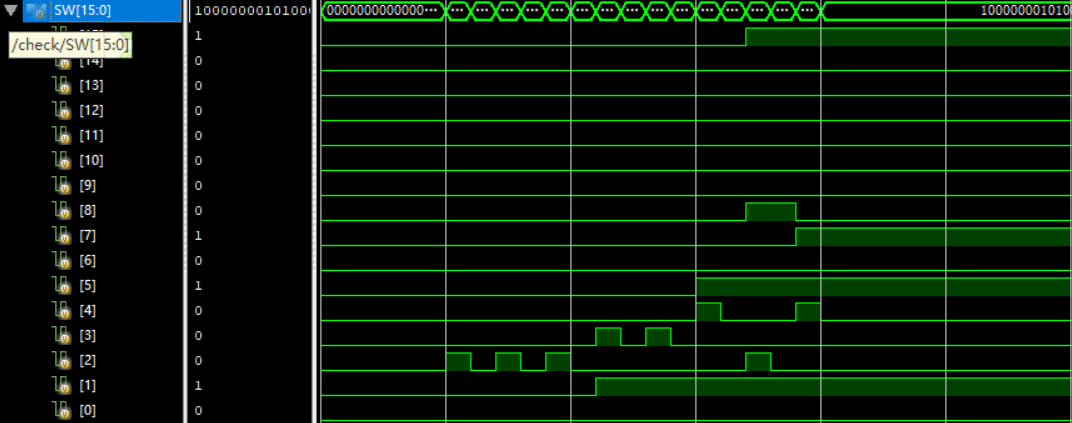
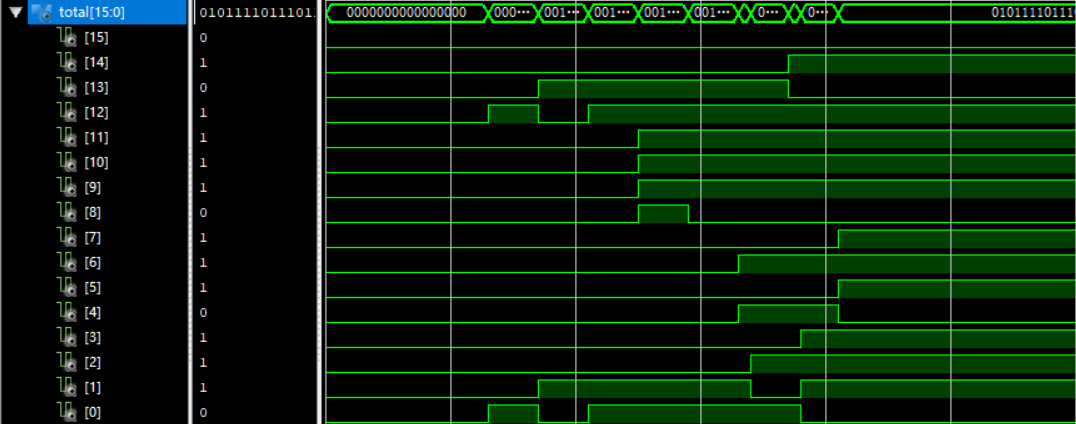
SW[8:7]=2'b01;

SW[4]=1;#10;

SW[4]=0;#10;

end

仿真结果波形图如下：

****

在本次仿真实验中，我们在第一次访问寄存器时，将寄存器的初值赋为4’b0000，拨动 SW[15] 至0，同时将 SW[0] 置为0，表示进行自增操作；通过上下拨动 SW[2] 开关，更新寄存器 A 中的数值，由波形图中可以看出，寄存器中数值不断进行 +1 操作，BUS 默认输出也随寄存器 A 的变化而变化；此时，将 SW[1] 置为 1，表示对寄存器 B 作自减操作，拨动开关 SW[3]，可以从波形图见到，寄存器 B 终数值不断自减，BUS 中数值不发生变化。

此时将 SW[6:5] 置为 01，表示选择寄存器 A 与 B 中数据进行减操作，并将通过拨动 SW[4] 使结果赋予寄存器 C 中，由 A = 3，B = E 的初始值可以得出，两个操作数作差的结果为 -11，其补码表示为 0101 显示为数字 5（此时进位为 1，表示减法存在借位的情况），波形图对此进行了验证。

此时调整 SW[8:7] 为 10，选择寄存器 C 中数据，保持 SW[15] 的开合状态，拨动 SW[2]，使 BUS 中数值赋予寄存器 A 中，从波形图中可以见到，随着loada 信号的产生，寄存器 A 中数值确实等于了寄存器 C 中数值；同理，将 SW[8:7] 置为 01，使 BUS 选取寄存器 B 中数值，通过拨动 SW[4] 将其赋予寄存器 C 中，通过波形图我们可以见到，实际的仿真符合这一操作。

由上述分析可以看出，我们的代码设计符合实现要求，说明程序设计的正确性。

**五、实验心得**

疫情的无常，我们不得不采用线上仿真的方式，来完成剩下的数逻实验；实话说，虽然线上仿真可以让我们更深入的了解 Verilog 语言，更透彻的解析特定功能的设计与实现，但是不能进行上板验证的弊端，会使数逻实验的过程减少很多的满足感与成就感，真是可惜。

在本次 lab 转为线上实验之前，我已经结合 PPT 的教学与自己的理解，总体对本实验作了一定的代码分析，但是编写代码的过程十分艰辛，还出现了很多难以理解的报错，就算最后实现了计数与数据传输功能，但也实现得不那么漂亮，且对一些按钮或开关的功能有所简化；但在研读了本次 lab 的线上版后，我发现计数和数据传输的功能实现都变得明晰起来，只需要记录 BUS 数据及自增/自减数据即可，通过 assign 语句就可以解决选择的问题，可谓是清晰而简单，真的是柳暗花明又一村。

虽然本次的实验形式发生了改变，但我的收获可谓是反增不减，真是精彩！



**本科实验报告**

**课程名称： 数字逻辑电路设计**

**姓 名：**

**学 院： 计算机科学与技术学院**

**专 业： 计算机科学与技术**

**邮 箱：**

**QQ 号：**

**电 话：**

**指导教师：**

**报告日期： 2022 年 12 月 17 日**

**浙江大学实验报告**

课程名称： 数字逻辑设计 实验类型： 综合

实验项目名称： 计时器、定时器的设计与应用

学生姓名： 学号： 同组学生姓名：

实验地点： 线上实验 实验日期： 2022 年 12 月 17 日

# **一、实验目的和要求**

# 1.掌握同步四位二进制计数器74LS161的工作原理和设计方法

# 2.掌握时钟/定时器的工作原理与设计方法

# **二、实验内容和原理**

**2.1 实验任务**

1.采用行为描述设计同步四位二进制计数器74LS161

2.基于74LS161设计时钟应用

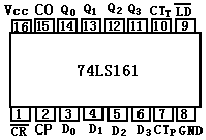
**2.2 实验原理**

2.2.1 同步四位二进制计数器74LS161

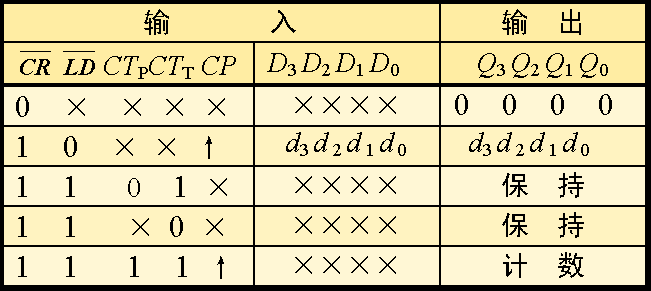
74LS161是常用的四位二进制可预置的同步加法计数器，其可灵活运用在各种数字电路，实现分频器等很多重要的功能。

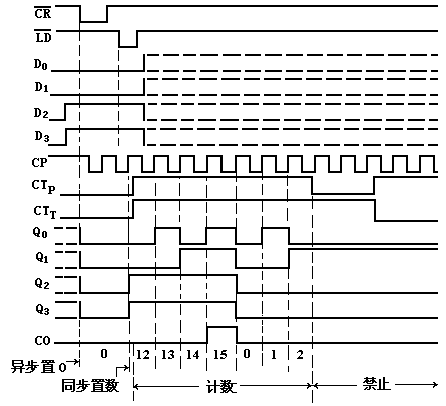
74LS161功能描述（见下示意图）：

其中 CRbar 为清零端，LDbar 为置数端，CTp、CTt 为使能端，C0 为进位输出端。Q1/Q2/Q3/Q4 为状态输出，D1/D2/D3/D4 为并行输入，CP 表示时钟输入。



74LS161 功能描述的对映真值表与时序表达图为：





由上两图可见，仅当 CRbar,LDbar，CTt，CPt 均为 1 时，时钟的上升沿到来会使 74LS161 模块开始计数；而当 CRbar 为 0 时，模块将输出状态置 0 ；当 CRbar,LDbar 分别为 1 和 0 时，当时钟的上升沿到来时，模块状态输出等于输入值；其他信号输入时，输出均保持不变。

2.2.2 实现十进制计数器

十进制计数器对映的真值表将转换为如下：



0 0 0 0 0

1 0 0 0 1

0 1 0 0 2

1 1 0 0 3

0 0 1 0 4

1 0 1 0 5

0 1 1 0 6

1 1 1 0 7

0 0 0 1 8

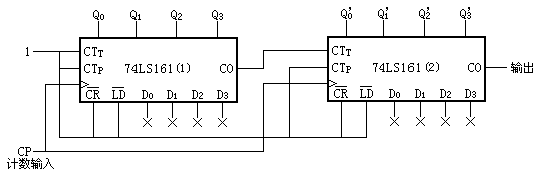
1 0 0 1 9

0 1 0 1 10->0

利用与非门拾取状态 1010 作用于清零端 CRbar，可以实现十进制计数（ 0000 到 1001 ），改变与非门的输入信号，则可以实现其它进制计数。

2.2.3 实现16×16进制计数器（同理）

采用两个 74LS161 模块实现，一个作为低位片，一个作为高位片，在计数计到 1111 以前，CO1 ＝ 0，高位片保持原状态不变，在计数计到 1111 时，CO1 ＝ 1使，高位片在下一个 CP 实现加一。



2.2.4 实现50进制计数器（16进制）

同理使用两个 74LS161 模块，十进制数 50 对应的二进制数为 0011 0010。根据低位片的当前输出状态控制进位情况，当第四位与第一位第一次同时为 1 时，表示十进制数字 9 的出现，此时置零低位片，并向高位片提供进位信号，使其计数加一，同理以 5 的二进制表示设置高位的清零状态。

2.2.5 分钟60进制（十进制显示）

设计原理同上，具体逻辑电路实现原理图如下：

数字时钟：

设计一个数字钟，使用 60 进制和 24 进制计数器，实现 24 小时内时间的实时显示。

数字钟的初值通过初始化语句来实现，用数码管前两位显示小时的十位和个位，后两位显示分钟的十位和个位。

**三、主要仪器设备**

1. 装有 Xilinx ISE 14.7 的计算机 1台

2. SWORD 开发板 1套

**四、操作方法与实验步骤**

**4.1 采用行为描述设计同步四位二进制计数器 74LS161**

新建工程，工程名称用为 clockcounter ；

Top Level Source Type 选择使用 HDL；

用行为描述设计，其中 CR 表示异步清零，LD 表示同步置位。

具体 verilog 代码如下：

module clockcounter(

input wire CP,

input wire CR,

input wire ld,

input wire CTT,

input wire CPT,

input wire [3:0] D,

output reg [3:0] Q,

output wire C0

);

initial begin

Q=0;

end

always@(posedge CP or negedge CR)begin

if(~CR)begin

Q<=0;

end

else if(~ld)begin

Q<=D;

end

else if(CTT & CPT)begin

Q<=Q+1;

end

end

assign C0=(&Q)&CP;

endmodule

为配合后续时钟的设计，在每次调用此模块时先将输出 Q 置 0 ，再根据真值表设计，分 if-else 语句实现赋值、数值保持或计数功能，此计数器为满 16 进 1，因为 Q 的设定只有四位，因而可以实现循坏计数，不会有溢出的产生。

仿真代码如下：

initial forever begin

CP=1;#5;

CP=0;#5;

end

initial begin

CR = 0;

D=0;

ld = 0;

CTT = 0;

CPT = 0;

D = 0;

#100;

CR=1;ld=1;

D=4'b1100;

CTT=0;CPT=0;

#30 CR=0;

#20 CR=1;

#10 ld=0;

#30 CTT=1;CPT=1;

#10 ld=1;

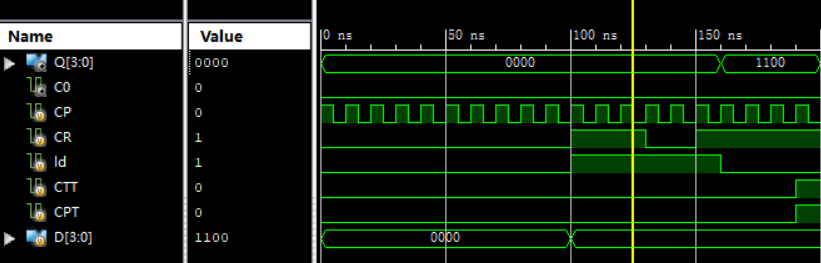
#510;CR=0;

#20 CR=1;

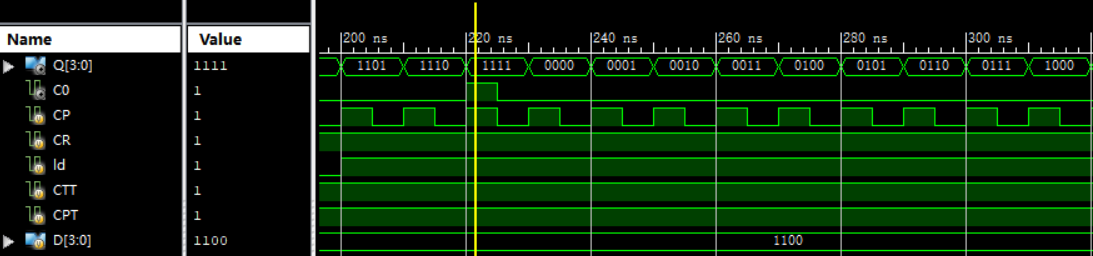
#500;

end

具体的仿真结果如下：



由上图可见，当 CR 为 0 时，输出被置 0，当 CR 为 1，ld 为 0 时，输出被赋予输入的值;当 CTT/CPT 为 0 时，即使 CR/ld 为任意组合时，计数均不会发生。



由上图结果可见，当 CTT/CPT 为 1，且 CR、ld 也同时为 1 时，输出开始计数表达，随着 clk（CP） 的到来，每次增加 1，增加至 1111 时，进位信号置为 1 ，表示进位的产生；且下一时钟信号到来时，输出被置为 0 ，模块仿真符合我们的设计功能要求。

**4.2 基于 74LS161 设计时钟应用**

4.2.1 新建工程，工程名称为 application；

Top Level Source Type 选择使用 HDL；

用结构化描述设计，其中调用模块 clockcounter，调用分频模块，用 100ms 作为分的驱动时钟（线上仿真时不使用，易使时间花费过大），调用显示模块（线上仿真不使用）。

设计 60 进制计数器，其 verilog 代码具体如下：

module application(

input wire clk,

output wire [7:0] num

);

//wire [31:0] clk\_100;

wire [1:0] C;

//clkdiv s0 (clk,clk\_100);

clockcounter m1(.CP(clk),.CR(1'b1),.ld(~(num[3] & num[0])),

.CTT(1'b1),.CPT(1'b1),.D(4'b0000),.Q(num[3:0]),.C0(C[0]));

clockcounter m2(.CP(clk),.CR(1'b1),

.ld(~(num[3] & num[0] & num[6] & num[4])),

.CTT(num[3] & num[0]),.CPT(1'b1),.D(4'b0000),

.Q(num[7:4]),.C0(C[1]));

endmodule

/\*module clkdiv(

input clk,

output reg [31:0]clkdiv

);

always@(posedge clk)begin

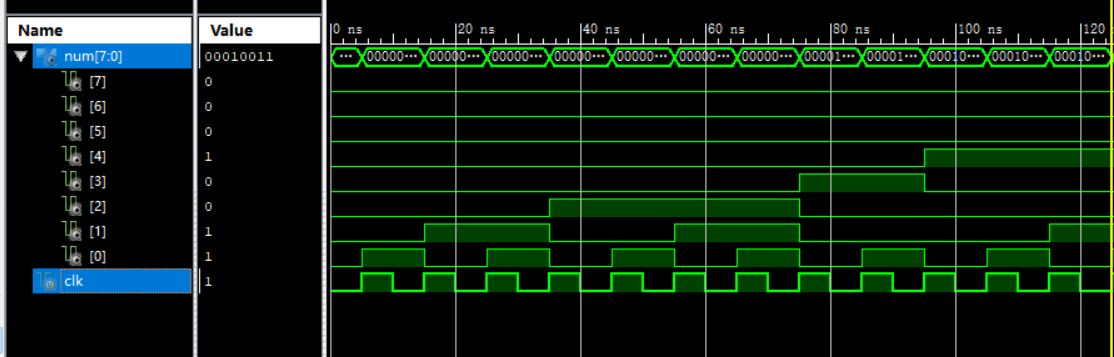
clkdiv<=clkdiv+1'b1;

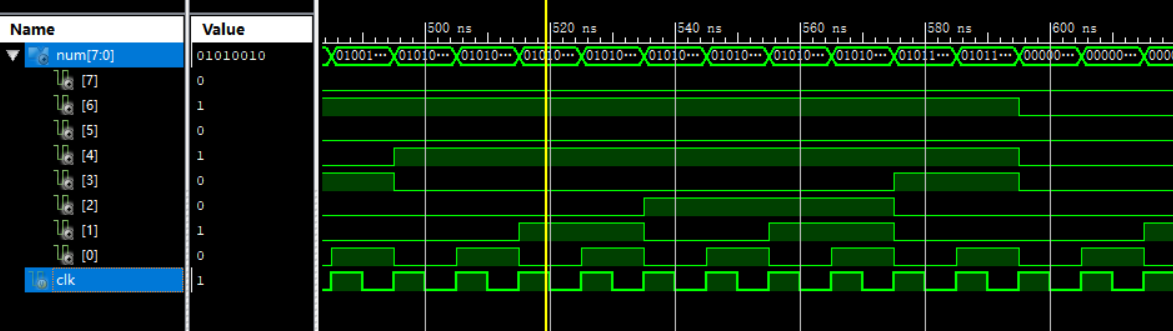
end

endmodule\*/

如前所述，通过输出的第三位与第零位控制 ld （置零，因为输入保持为 0000，而 CR 为 1，ld 为 0 时，输入赋值于输出）信号的产生，使低位片重新计数，并将进位信号传递给高位片，高位片逢 6 进 1 的处理同理。

可以通过 clk 的持续变化进行仿真，仿真结果如下：



由上图可见，clockcounter 模块调用正常，可以进行 0-9（十进制）的计数，且每次到达 9 后，随着下一个时钟周期的到来，低位片被清零，高位片增加 1，符合一般设计需求。

当模块增加至 59（低位片为 9，高位片为 5）后，随着下一个时钟周期的到来，低位片、高位片均被置位为 0，进而开始重新计数，因而符合我们的设计需求。

4.2.1 实现 24 小时的时钟记录

新建工程，工程名称为 topclock；

Top Level Source Type 选择使用 HDL；

用结构化描述设计，其中调用模块 clockcounter，调用分频模块，用 100ms 作为分的驱动时钟（线上仿真时不使用，易使时间花费过大），调用显示模块（线上仿真不使用）。

具体 verilog 代码实现如下：

module topclock(

input wire clk,

output wire [23:0]num

);

//wire [31:0] clk\_100;

wire [5:0] C;

wire [5:0] abc;

//clkdiv s0 (clk,clk\_100);

//秒计数

clockcounter m1(.CP(clk),.CR(1'b1),.ld(~abc[0]),.CTT(1'b1),

.CPT(1'b1),.D(4'b0000),.Q(num[3:0]),.C0(C[0]));

clockcounter m2(.CP(clk),.CR(1'b1),.ld(~abc[1]),.CTT(abc[0]),

.CPT(1'b1),.D(4'b0000),.Q(num[7:4]),.C0(C[1]));

assign abc[0]= num[3] & num[0];

assign abc[1]= abc[0] & num[6] & num[4];

//分计数

clockcounter m3(.CP(clk),.CR(1'b1),.ld(~abc[2]),.CTT(abc[1]),

.CPT(1'b1),.D(4'b0000),.Q(num[11:8]),.C0(C[2]));

clockcounter m4(.CP(clk),.CR(1'b1),.ld(~abc[3]),.CTT(abc[2]),

.CPT(1'b1),.D(4'b0000),.Q(num[15:12]),.C0(C[3]));

assign abc[2]= abc[1] & num[11] & num[8];

assign abc[3]= abc[2] & num[14] & num[12];

//时计数

clockcounter m5(.CP(clk),.CR(1'b1),.ld(~abc[4]),.CTT(abc[3]),

.CPT(1'b1),.D(4'b0000),.Q(num[19:16]),.C0(C[4]));

clockcounter m6(.CP(clk),.CR(1'b1),.ld(~abc[5]),.CTT(abc[4]),

.CPT(1'b1),.D(4'b0000),.Q(num[23:20]),.C0(C[5]));

assign abc[4]= abc[3] & ((num[19] & num[16])|

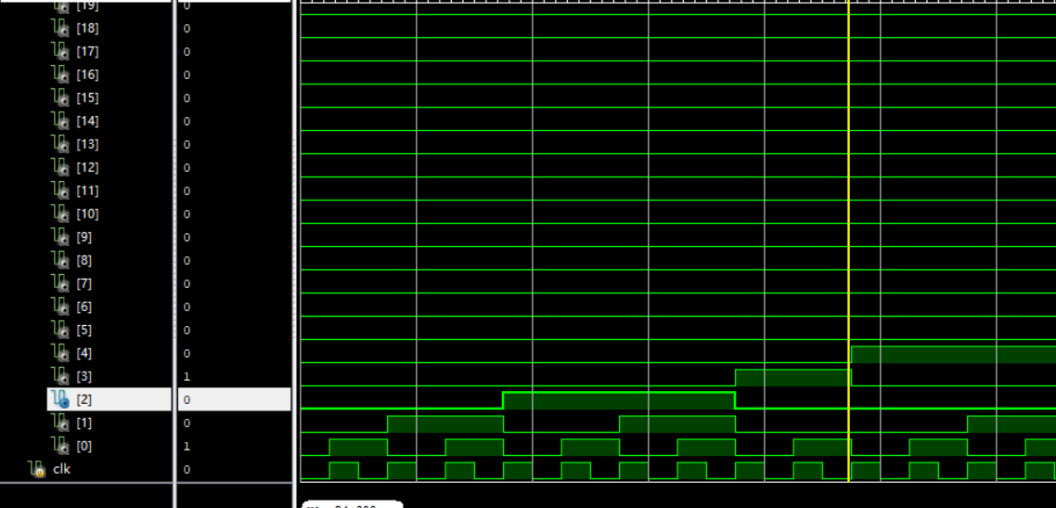
(num[17] & num[16] & num[21]));

assign abc[5]= abc[4] & num[21];

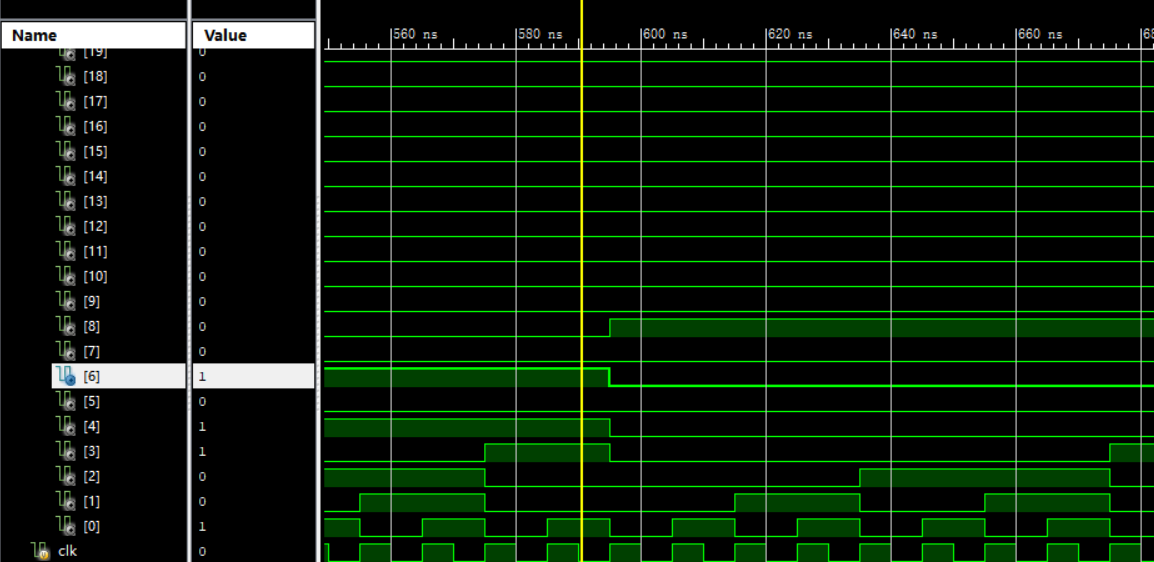
endmodule

其中的 assign 语句的单独设立仅为了简便 clockcounter 模块中的操作条件；其中“秒”和“分”的计数使用前述 59（十进制）进位的设计思想进行信号筛选，其中“秒”的置零判断同前述设计，“分”的置零判断需要综合“秒”的进位，是“与”操作的叠加过程；而对于最后的“时”而言，需要在逢 9 进一的基础上，“或”操作逢 4 进一的可能性（仅当“时”的高位片为 2 时提供筛选的可能），而“时”的高位片逢 3 进一的设计则与“分”“秒”中逢 6 进一的设计相同。

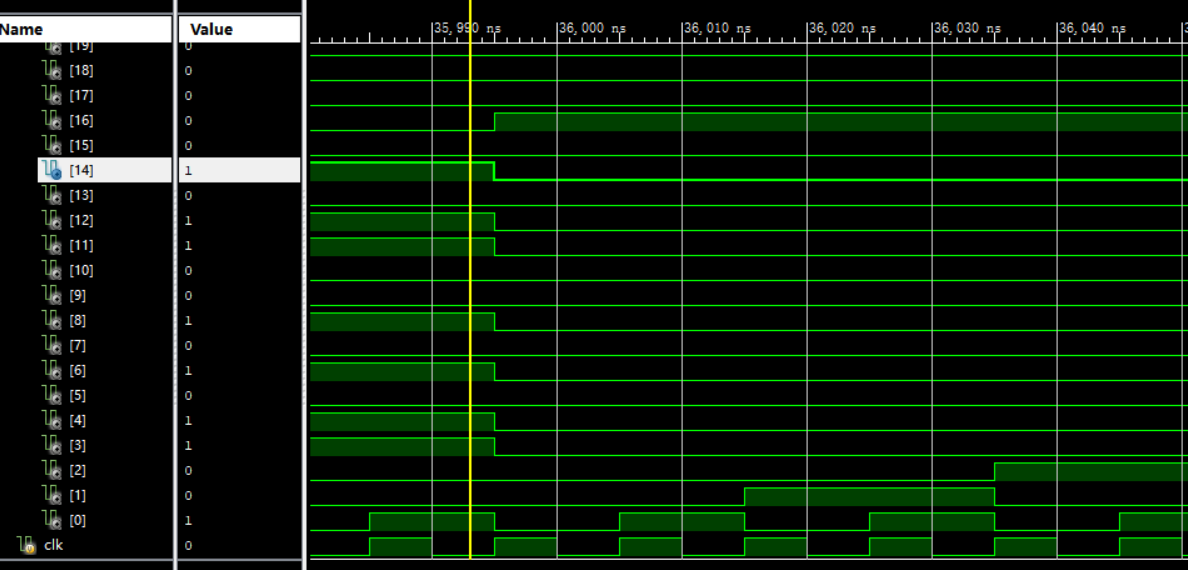
可以通过 clk 时钟的不断 01 变化进行仿真，仿真结果如下：

****

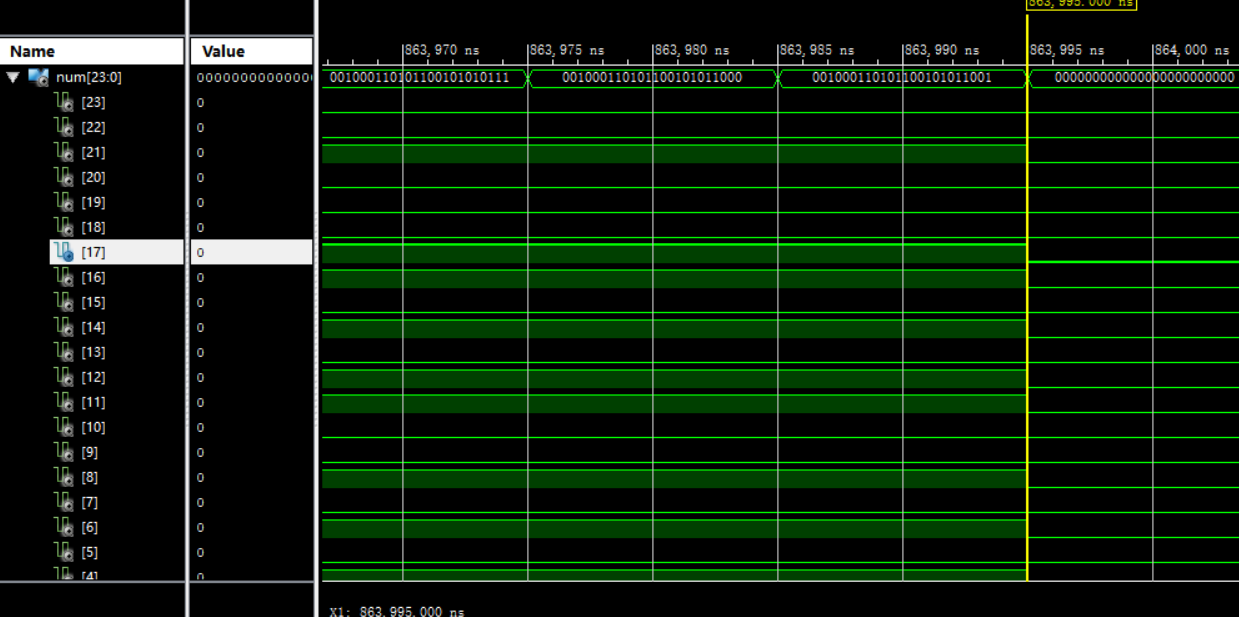
由上图可见，随着 clk（CP）时钟的到来，“秒”为正常的进行计数操作，且低位片逢 9 进 1，引起高位片的增加。



由上图可见，随着计数的增加，当 num[7:4] 增加到 0101（二进制）时,num[3:0] 增加到 1001（二进制）时，表示“秒”位已计数到 59 ，随着下一个时钟周期的到来，“秒”位的低位片与高位片均被置 0，且 num[8] 增加 1，表示 59 秒的计数完成从而产生分的进位，符合我们的设计预期，说明该模块在“秒”位与“分”位的衔接中进位正常。

****

由上图可见，随着“分”位计数的上升，num[15:8] 的二进制表达数也不断的增大,因为使用的“分”进位模块与先前“秒”的相统一，所以在此就不再过多的说明 num[15:8] 内部是如何逢 9 进 1 的；当“分”和“秒”同时来到 59时，意味着 59 分 59 秒，随着下一个时钟周期的到来，“时”位需要进一，由图中仿真结果可知，num[16] 被置为 1，而其余的 num[15:0] 置位为 0，功能实现符合预期，说明“时”与“分”“秒”之间的衔接正确。

****

我们最后关注，当计时器来到 23 时 59 分 59 秒，随着下一个时钟周期的到来，会发生什么样的变化；因为在之前的设计中，已经通过“或”操作时“时”的进位既是逢 9 进 1，又在特定条件下，可以实现逢 4 进 1，因而在仿真的结果图中我们看到，二进制数从 00100011 01011001 01011001 转变为 00000000 00000000 00000000，意味着时间的重置（置零），之后随着时钟周期的到来，又开始重新一轮的计数过程；因而符合我们的设计需求，说明时钟计数器模块的设计正确性。

**五、实验心得**

第二个线上实验，在“小摆”了几个礼拜后，终于连上了学校的 ftp，开始新一轮的数逻实验进程。

经过了上一轮寄存器实验的摧残，我对于模块附初值有了比较系统的认识，因而在本次实验的最终 top （时钟计数器）模块中，较为迅速的 cover 了 num变量没有初值就需要加以判断的问题，真的是学以致用。

本次计数器的实验过程，整体实现并不困难，各部分模块的设计过程以及最后 top 部分的整合，逻辑及实现思路都十分清晰，虽然离开了“保姆级”的 PPT教程，但随着对 verilog 语言的深入学习，感觉一切都逐渐变得得心应手起来；这次实验的关键在于厘清每一个位片的输入条件，特别是什么时候置零的问题，以及在当前高位片与下一个低位片之间的，置零信号如何传递的具体实现问题，设计较为困难的部分可能是“时”位模块，其既要符合逢 9 进 1 的一般特性，又要在一定的时机，符合逢 4 进 1 的特性，而这样的数学逻辑构想刚好与“或”们的操作相统一，因而我最后也是用“或”操作很好的实现了它。

虽是清晰简洁的一次实验，但在其应用性的体验上，真是受益匪浅。



**本科实验报告**

**课程名称： 数字逻辑电路设计**

**姓 名：**

**学 院： 计算机科学与技术学院**

**专 业： 计算机科学与技术**

**邮 箱：**

**QQ 号：**

**电 话：**

**指导教师：**

**报告日期： 2022 年 12 月 25 日**

**浙江大学实验报告**

课程名称： 数字逻辑设计 实验类型： 综合

实验项目名称： 移位寄存器设计与应用

学生姓名： 学号： 同组学生姓名：

实验地点： 线上实验 实验日期： 2022 年 12 月 25 日

# **一、实验目的和要求**

# 1.掌握支持并行输入的移位寄存器的工作原理

# 2.掌握支持并行输入的移位寄存器的设计方法

# **二、实验内容和原理**

**2.1 实验任务**

1.设计8位带并行输入的右移移位寄存器

2.设计跑马灯应用

**2.2 实验原理**

2.2.1 移位寄存器

每来一个时钟脉冲，寄存器中的数据按顺序向左或向右移动一位；因而必须采用主从触发器或边沿触发器，不能采用锁存器。

数据移动方式：左移、右移、循环移位。

数据输入输出方式：

1.串行输入，串行输出

2.串行输入，并行输出

3.并行输入，串行输出

2.2.2 串行输入右移移位寄存器

使用D触发器构成串行输入的右移移位寄存器，其电路逻辑示意图如下：



2.2.3 循环右移移位寄存器

在串行输入移位寄存器的基础上，将溢出位充当串行输入的数值，其逻辑电路示意图如下：



2.2.4 带并行输入的右移移位寄存器

数据输入方式：串行输入、并行输入，其逻辑电路示意讲解图如下：



**三、主要仪器设备**

1. 装有 Xilinx ISE 14.7 的计算机 1台

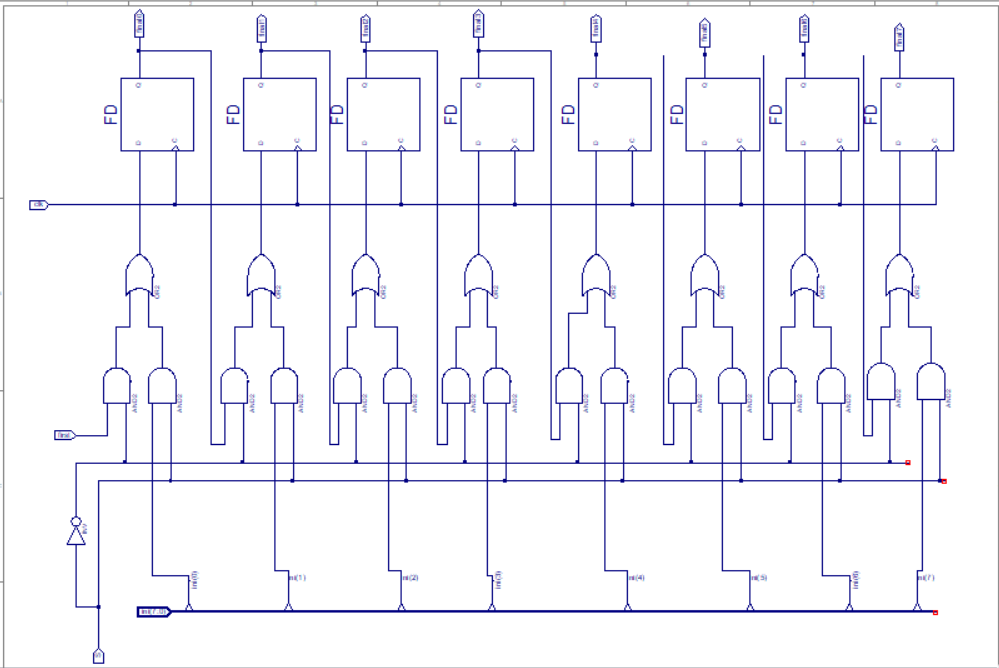
2. SWORD 开发板 1套

**四、操作方法与实验步骤**

4.1 设计8位带并行输入的右移移位寄存器

1.新建工程，工程名称为 shift\_reg, Top Level Source Type 选择使用 HDL，

2.采用绘图方式给出设计，逻辑电路设计图如下：



3.对设计结果进行仿真检验

具体的仿真代码如下：

initial forever begin

clk=0;#10;

clk=1;#10;

end

initial begin

S = 0;first = 0;ini = 0;#100;

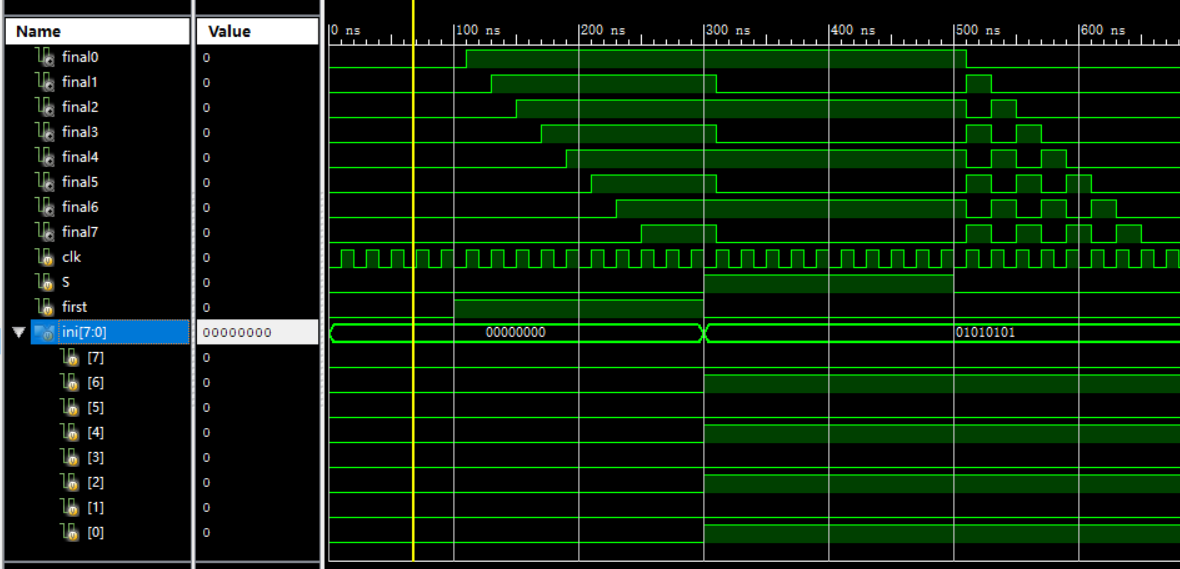
S=0;first=1;ini=0;#200;

S=1;first=0;ini=8'b0101\_0101;#200;

S=0；#200；

end

仿真结果如下：



由仿真结果我们可以看到，当一般输入数值为 0 时，输出保持初始状态“不变”（其实是 0 的不断右移），若将一般输入置为 1 后，可见在载入位为 0 时，数值 1 不断进行右移扩展，直至数值为 1111\_1111，若将载入位置 1 ，此时可见输出的数值等于输入数值，且因为载入位的持续保持，输出结果不发生变化；当载入位重新置位为 0 后，可见模块又开始进行右移操作，因而该模块的设计符合我们功能的需要。

4.2 设计跑马灯应用

1.新建工程，工程名称使用 marqueue， Top Level Source Type 选择使用 HDL；

2.用结构化描述设计，其中调用 ShfitReg8b、分频模块（用1s作为移位寄存器驱动时钟）、显示模块、CreateNumber 模块；（线上仿真部分省略）

具体设计 verilog 代码如下：

module newqueue(

input wire [4:0] SW,

input wire clk,

output wire [7:0]num

);

wire [3:0] A,B;

myregister4b a1(.clk(clk),.IN(A),.OUT(A),.check(SW[0]));

myregister4b b1(.clk(clk),.IN(B),.OUT(B),.check(SW[1]));

newtry m1(.clk(clk),.firsti(SW[3]),.selectli(SW[4]),.D({A,B}),

.start(SW[2]),.ledout(num[7:0]));

endmodule

module myregister4b(

input wire clk,

input wire [3:0] IN,

input wire check,

output reg [3:0] OUT

);

initial OUT=4'b0000;

always@(posedge clk)begin

if(check) OUT<=IN+1;

else OUT<=IN;

end

endmodule

其对映的仿真代码如下：

initial forever begin

clk=1;#5;

clk=0;#5;

end

initial begin

SW=0;#100;

#20;

SW[3]=1;#20;

SW[3]=0;#20;

SW[3]=1;#20;

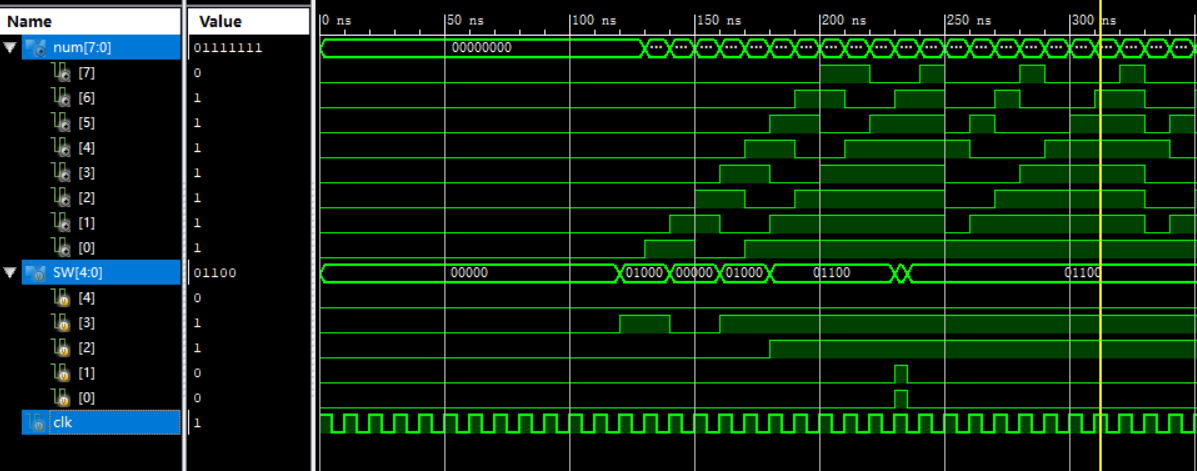
SW[2]=1;#50;

SW[0]=1;SW[1]=1;#5;

SW[0]=0;SW[1]=0;#200;

SW[2]=0;SW[4]=1;#100;

end

仿真结果如下：

由仿真结果可以见到，初始状态下输出保持 0000\_0000 的状态（其实是 0 的不断右移，与输出初始状态重合），若仅改变一般输入，则右移补充项转变为数值 1 的传递；可见一般输入数值在 0、1 间转换的输入功能有效；

此时打开开关 2，调节一般输入转变为指定输入，经过一轮倒计时后（初值跳马灯预告信号），输出值被置为寄存器 A、B 中的保留数值输入，在本实验中，我将寄存器 A、B 初始化为 0 ，但在输入之前，分别打开一个时钟周期的开关 0 与 1 ，因而可见 A、B 寄存器中的数值均转变为 0001，说明开关 0、1 使寄存器中数值增加及开关 2 使寄存器数值的载入功能实现正常；

最后，打开开关 4，调节模块功能为循环输入（原始为串行输入），此时可以看到，在整个右移的过程中，输出值的最后一位重新作为输入输入到输出值中，因而符合我们循环输入的功能需求；

通过仿真结果与以上的分析，可见我们跑马灯应用模块的设计得以实现。

**BONUS 1**

设计 8 位带并行输入的移移位寄存器

（感觉和前面的一般设计有点重复，我便稍微拓展了一点---根据 PPT 做了七位串并行转换器）

设计原理：

当没有启动命令时，模块一直处于一般输入的串行传递状态，其电路逻辑设计示意图如下：



当启动命令来到时时，模块跳出一般输入的串行传递状态，通过外来输入对输出进行赋值，其电路逻辑设计示意图如下：



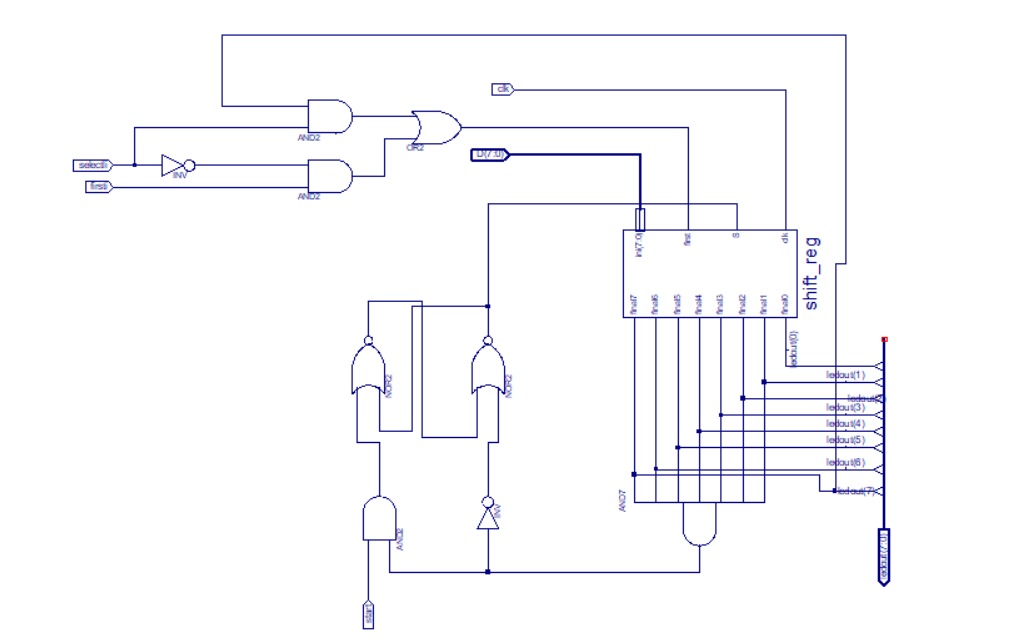
总结：并行－串行转换器工作过程

并行－串行转换器设计原理：当没有启动命令（低电平）时，电路上电后经过若干个时钟脉冲后将会稳定在 RS 触发器输出 q ＝ 0，移位寄存器 Q7 - Q0＝ 11111111 的状态。

当启动命令（高电平）加至启动输入端时，RS 触发器的输出端 q 被置 1，7位并行数据及标志码“0”在第一个 clk 的作用下同时置入移位寄存器。  
 此时，由于 P7 = 0 导致 Q7 = 0，使得七输入与门的输出变成 0，一方面封锁启动命令的输入，另一方面通过非门在 RS 的触发器的 r 端输入 1，使 RS触发器的输出 q = 0 ，移位寄存器进入移位状态。

再在时钟脉冲作用下，一方面使并行数据串行移出，另一方面又不断将“ 1 ”移入寄存器。等第 7 个脉冲来到后，七输入与门的输入已全为“ 1 ”，使得其输出变为 1，标志着转换完成，同时解除对启动信号的封锁，当再来一个启动命令时又可以再次进行并行－串行转换

具体设计如下：



采用以下代码进行仿真：

initial forever begin

clk=1;#10;

clk=0;#10;

end

initial begin

D=8'b00110011;start=0;selectli=0;firsti=1;#200;

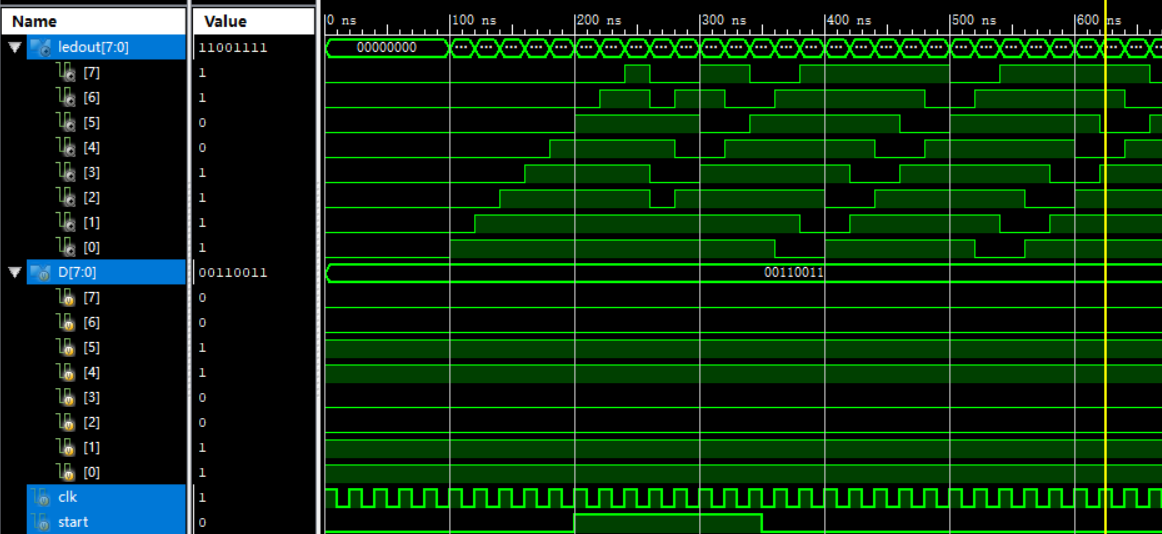
start=1;#150;

start=0;selectli=1;#200;

end

endmodule

仿真结果如图：

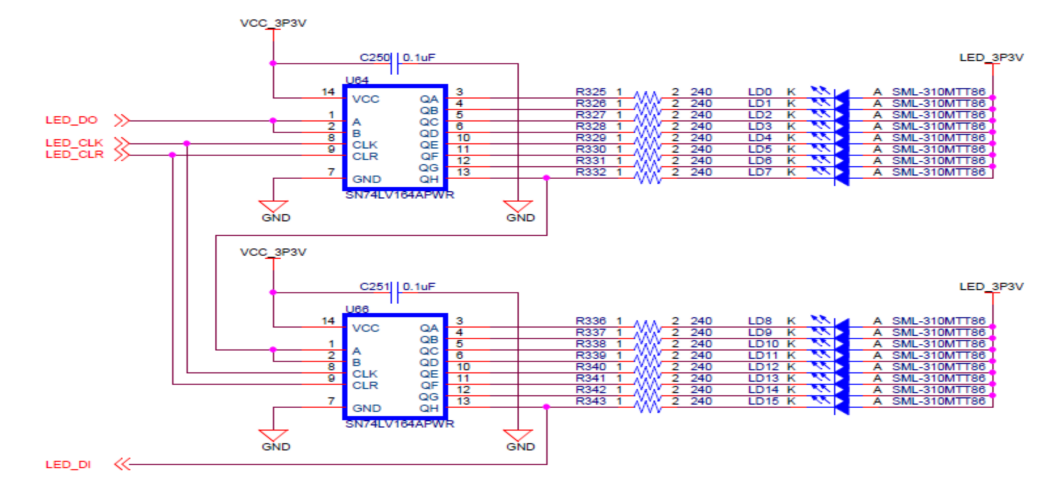


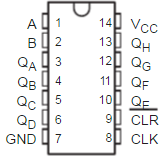
由仿真结果可以看到，一般输入的右移功能、指定输入的载入功能以及一般输入的数值变化传递等要求，本模块都可以正常的实现，因而设计具有一定的正确性。

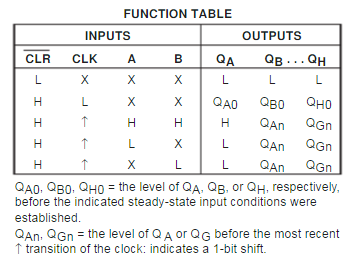
**BONUS 2**

设计主板 LED 灯驱动模块

接口说明：

主板 LED 灯，采用 2 个 74LV164A 构成 16 位串行输入并行输出移位寄存器，并行输出控制 16 个 LED 灯。

其中 74LV164A 模块说明与对映真值表如下：





LED 并行显示模块 M6：SPIO

1.15 位 LED 指示灯控制(IP Core)：

1.1逻辑实验的输出LED显示模块

1.1.1相当于通用输入输出接口：GPIO；

1.1.215 位用于 LED 指示控制，其余用于扩展；

1.2器件编号改为U7；

1.3本课程用于调试显示和CPU的简单外设；

2.基本功能：

2.1输入32位二进制数据：P\_Data

clk：时钟，EN：输出使能，Start：串行扫描启动，rst：复位

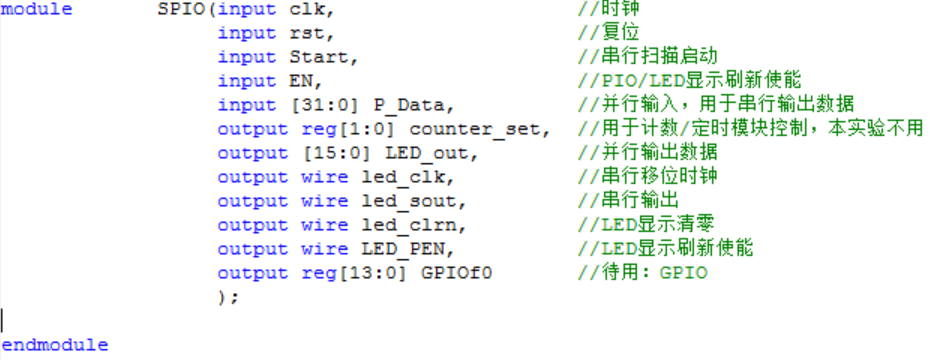
2.2串行输出：led\_clk = 时钟，led\_sout = 串行输出数据，LED\_PEN = 使能，led\_clrn = 清零

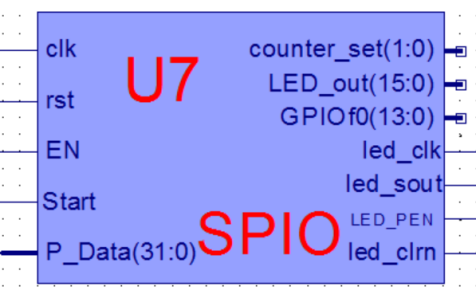
2.3并行输出：LED\_out、counter\_set、GPIOf0

3.核模块符号文档：SPIO.sym

LED并行显示模块IP核端口信号

PIO/LED-GPIO IP核端口信号可作为IP核调用空文档：端口文档；具体设计说明如下：





任务：

1.新建工程，工程名称使用 LEDP2S，Top Level Source Type 选择使用 HDL

2.用结构化描述设计，设计中调用 CreatNumber 模块，用 4 位七段数码管设置16位LED灯的初值，调用显示模块，并改造 ShfitReg8b 模块，设计 LED 灯驱动模块 LED\_DRV.

具体设计 verilog 代码如下：

module LEDP2S(

input wire clk,

input wire [15:0] SW,

input wire [3:0] btn,

output wire [3:0] AN,

output wire [7:0] SEGMENT,

output wire LEDCLK,

output wire LEDCLR,

output reg LEDEN,

output wire LEDDT);

wire [15:0]num;

wire final;

wire [31:0] clkdiv;

clk\_div m1(.clk(clk),.rst(1'b0),.clkdiv(clkdiv));

shiftreg16b m2(.clk(clkdiv[25]),.load(SW[15]),.ini(num),

.outi(LEDDT),.final(final));

disp\_num m3(clk,num,4'b0,4'b0,1'b0,AN,SEGMENT);

creatnumber m4(btn[3:0],SW[3:0],num);

endmodule

module shiftreg16b(

input wire clk,

input wire load,

input wire [15:0] ini,

output reg outi,

output reg final

);

reg [4:0] count;

reg [15:0] midi;

always@(posedge clk)begin

if(load)begin

midi<=ini;

count<=4'b0000;

end

else begin

midi<={1'b1,midi[15:1]};

outi<=midi[0];

count<=count+1'b1;

if(count>=5'b10000)begin

final<=1'b0;

count<=5'b10000;

end

else begin

final<=1'b1;

end

end

end

endmodule

module clk\_div(

input clk,

input rst,

output reg [31:0] clkdiv

);

always@(posedge clk or posedge rst)begin

if(rst)clkdiv<=0;

else clkdiv<=clkdiv+1'b1;

end

endmodule

module creatnumber(

input wire [3:0] btn,

input wire [3:0] sw,

output reg [15:0] num

);

wire [3:0] A,B,C,D;

initial num<=16'b1010\_1011\_1100\_1101;

bit4addsub a1(.A(num[3:0]),.B(4'B0001),.fu(sw[1]),.S(A));

bit4addsub a2(.A(num[7:4]),.B(4'B0001),.fu(sw[0]),.S(B));

bit4addsub a3(.A(num[11:8]),.B(4'B0001),.fu(sw[3]),.S(C));

bit4addsub a4(.A(num[15:12]),.B(4'B0001),.fu(sw[2]),.S(D));

//增加数字的位置

always@(posedge btn[1]) num[3:0]<=A;

always@(posedge btn[0]) num[7:4]<=B;

always@(posedge btn[2]) num[11:8]<=C;

always@(posedge btn[3]) num[15:12]<=D;

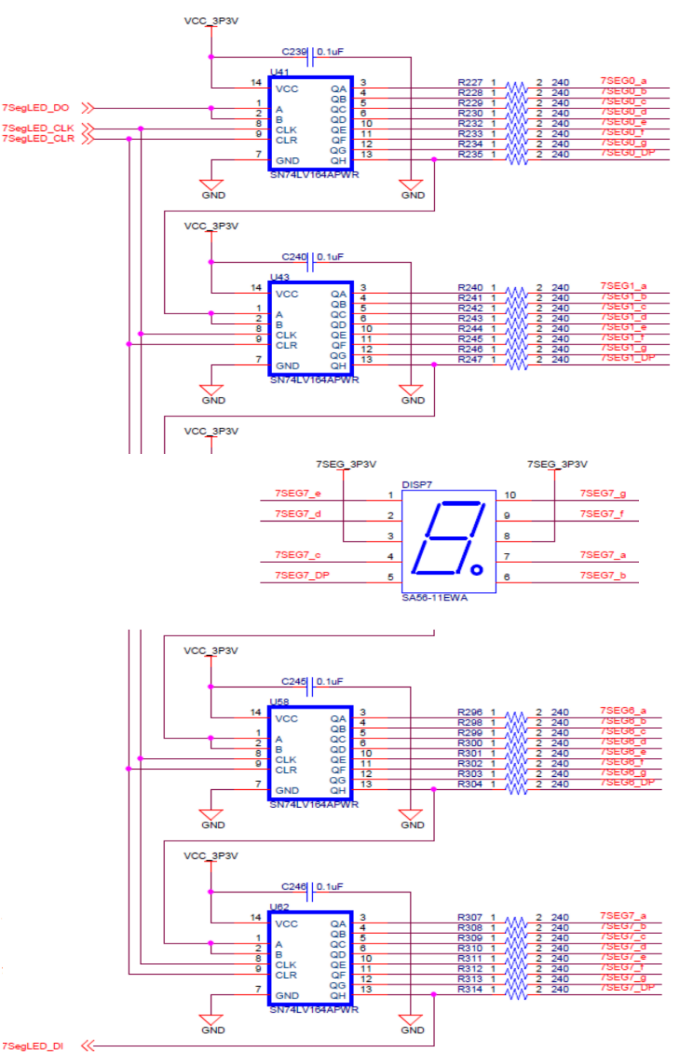
endmodule

**BONUS 3**

设计主板七段数码管驱动模块

设计原理：

静态译码74LS164移位输出，具体示意图如下：



七段码显示器IP核M3：SSeg7\_Dev

1.8位七段码显示器(IP Core)：

逻辑实验的输出显示模块；

本课程用于调试显示和 CPU 的简单外设；

器件编号改为 U6 ；

2.基本功能

2.1 输入 32 位二进制数据：Hexs；

2.1.1 SW[0] = 1，显示 8 位 16 进制数， SW[0] = 0，显示七段码 LED 点阵；

2.1.1.1 SW[0] = 1时：SW[1] = 1高 16 位，SW[1] = 0 低 16 位；

2.1.2 Flash 七码闪烁频率，由通用分频器 U8(Div[25]) 提供，Start 串行扫描启动，point:七段小数点，LES:七段码使能，闪烁指示；

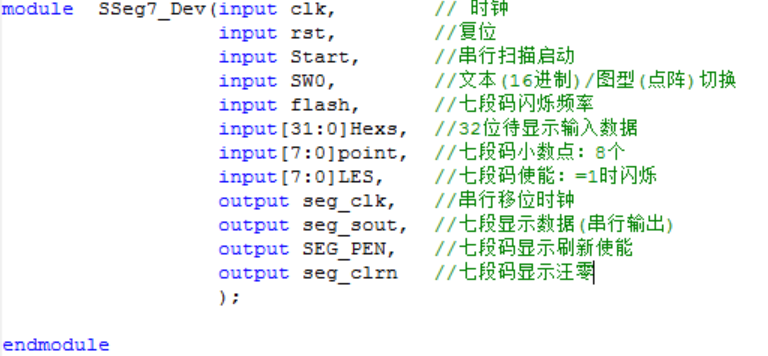
2.2 串行输出：seg\_clk = 时钟，seg\_out = 串行七段显示数据，SEG\_PEN =使能，seg\_clrn = 清零；

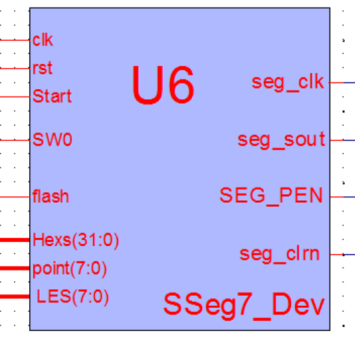
3.核模块符号文档：SSeg7\_Dev.sym

由实验二优化扩展，本实验提供 U6 的 IP 核。

七段码显示器IP核端口信号：

七段码显示器IP核端口信号可作为IP核调用空文档：端口文档；





任务：

1.新建工程，工程名称使用 SEGP2S，Top Level Source Type 选择使用 HDL

2.用结构化描述设计，改造 ShfitReg8b 模块，设计主板七段数码管驱动模块 SEG\_DRV，其中调用 MyMC14495 模块和显示模块。

具体设计 verilog 代码如下：

module SEGP2S(

input wire clk,

input wire [15:0]SW,

input wire [3:0]btn,

output wire SEGCLK,

output wire SEGCLR,

output wire SEGDT,

output wire SEGEN

);

wire [31:0] num;

wire [63:0] final;

wire [31:0] clkdiv;

wire number;

clk\_div m1(clk,1'b0,clkdiv);

creatnumber m2(btn[3:0],SW[3:0],num);

shiftreg64b m3(clkdiv[31],SW[15],final,SEGDT,number);

//将十进制数转化为七段数码管的各段

transfer t1(num[3:0],clkdiv[25],final[7:0]);

transfer t2(num[7:4],clkdiv[25],final[15:8]);

transfer t3(num[11:8],clkdiv[25],final[23:16]);

transfer t4(num[15:12],clkdiv[25],final[31:24]);

transfer t5(num[19:16],clkdiv[25],final[39:32]);

transfer t6(num[23:20],clkdiv[25],final[47:40]);

transfer t7(num[27:24],clkdiv[25],final[55:48]);

transfer t8(num[31:28],clkdiv[25],final[63:56]);

assign SEFCLK =clkdiv[25]&number;

assign SEGCLR = 1'b1;

assign SEGEN = 1'b1;

endmodule

module shiftreg64b(

input wire clk,

input wire load,

input wire [63:0] ini,

output reg outi,

output reg final

);

reg [6:0] count;

reg [63:0] midi;

always@(posedge clk)begin

if(load)begin

midi<=ini;

count<=7'b0000\_000;

end

else begin

midi<={1'b1,midi[63:1]};

outi<=midi[0];

count<=count+1'b1;

if(count>=7'b1000\_000)begin

final<=1'b0;

count<=7'b1000\_000;

end

else begin

final<=1'b1;

end

end

end

endmodule

module transfer(

input wire [3:0]num,

input wire clk,

output reg [7:0] final );

always@(posedge clk)begin

case(num[3:0])

4'b0000:begin final[7:0]<=8'b0000\_0010;end

4'b0001:begin final[7:0]<=8'b1001\_1110;end

4'b0010:begin final[7:0]<=8'b0010\_0100;end

4'b0011:begin final[7:0]<=8'b0000\_1100;end

4'b0100:begin final[7:0]<=8'b1001\_1000;end

4'b0101:begin final[7:0]<=8'b0100\_1000;end

4'b0110:begin final[7:0]<=8'b0100\_0000;end

4'b0111:begin final[7:0]<=8'b0001\_1110;end

4'b1000:begin final[7:0]<=8'b0000\_0000;end

4'b1001:begin final[7:0]<=8'b0000\_1000;end

4'b1010:begin final[7:0]<=8'b0001\_0000;end

4'b1011:begin final[7:0]<=8'b1100\_0000;end

4'b1100:begin final[7:0]<=8'b0110\_0010;end

4'b1101:begin final[7:0]<=8'b1000\_0100;end

4'b1110:begin final[7:0]<=8'b0110\_0000;end

4'b1111:begin final[7:0]<=8'b0111\_0000;end

endcase

end

endmodule

module clk\_div(

input clk,

input rst,

output reg [31:0] clkdiv

);

always@(posedge clk or posedge rst)begin

if(rst)clkdiv<=0;

else clkdiv<=clkdiv+1'b1;

end

endmodule

module creatnumber(

input wire [3:0] btn,

input wire [3:0] sw,

output reg [15:0] num

);

wire [3:0] A,B,C,D;

initial num<=16'b1010\_1011\_1100\_1101;

bit4addsub a1(.A(num[3:0]),.B(4'B0001),.fu(sw[1]),.S(A));

bit4addsub a2(.A(num[7:4]),.B(4'B0001),.fu(sw[0]),.S(B));

bit4addsub a3(.A(num[11:8]),.B(4'B0001),.fu(sw[3]),.S(C));

bit4addsub a4(.A(num[15:12]),.B(4'B0001),.fu(sw[2]),.S(D));

//增加数字的位置

always@(posedge btn[1]) num[3:0]<=A;

always@(posedge btn[0]) num[7:4]<=B;

always@(posedge btn[2]) num[11:8]<=C;

always@(posedge btn[3]) num[15:12]<=D;

endmodule

**五、实验心得**

一晃之间，一个学期就那么过去了，而数逻实验的学习，也要就此告一段落；疫情无常，隔离加线上的实验生活，确乎不如线下操作来得明晰，或许有仿真激励代码的新要求，可以在一定程度上检验我们实验操作的设计效果，加深我们对 verilog 语言的理解，但线上实验带来的成就感与完成感，却真的太少。

没有上板的验证环节，设计“硬件”就好似另一种虚拟编程，我写的还是程序，只能想象我在虚拟中通过逻辑门，搭建了一个个新功能的模块。实验最后的两个 BONUS，可能因为线上自己研究的原因，我没有很清晰的洞察到它实验的内在要求，因而只能根据自己的理解，设计了两个模块，去模仿，去实践。