# 实验10—移位寄存器设计与应用报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名： | 赵冰骞 | | 学号： | 3090103420 | | | 专业： | 理科试验班 | | |
| 课程名称： | | 逻辑与计算机设计基础实验 | | | 同组学生姓名： | 朱里 | | | |
| 实验时间： | | 2010-12-10 | 实验地点： | | 紫金港东4-509 | | 指导老师： | | 王总辉 | |

# 一、实验目的和要求

1. 掌握移位寄存器的工作原理及设计方法
2. 掌握串、并数据转换的概念与方法
3. 了解序列信号在CPU控制器设计中的应用

# 二、实验内容和原理

## 2.1 带并行置入的移位寄存器

* 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
  + 必须采用主从触发器或边沿触发器
  + 不能采用电平触发器
* 数据移动方式：左移、右移
* 数据输入输出方式
  + 串行输入，串行输出
  + 串行输入，并行输出
  + 并行输入，串行输出

## 2.2 串行输入的移位寄存器

使用D触发器，可构成串行输入的移位寄存器



图表1 串行输入的移位寄存器

## 2.3 带并行置入的右移移位寄存器

* 数据输入移位寄存器的方式：串行输入、并行输入



图表2 带并行置入的右移移位寄存器

# 三、主要仪器设备

1. 装有ISE的计算机系统 1台
2. Spartan-III 开发板 1套

# 四、操作方法与实验步骤

## 4.1 设计一个16位带并行输入的右移移位寄存器

1. 编写该移位寄存器的代码模块shift\_reg

|  |
| --- |
| `timescale 1ns / 1ps  module shift\_reg(clk, S, s\_in, p\_in, Q);  input wire clk, S, s\_in;  input wire [15:0] p\_in;  output wire [15:0] Q;  wire [15:0] D;  wire nS;    shift\_reg\_8 m0(clk, S, s\_in, p\_in[15:8], Q[15:8]);  shift\_reg\_8 m1(clk, S, Q[8], p\_in[7:0], Q[7:0]);    endmodule  module shift\_reg\_8(clk, S, s\_in, p\_in, Q);  input wire clk, S, s\_in;  input wire [7:0] p\_in;  output wire [7:0] Q;  wire [7:0] D;  wire nS;  FD FDQ0(.C(clk), .D(D[0]), .Q(Q[0])),  FDQ1(.C(clk), .D(D[1]), .Q(Q[1])),  FDQ2(.C(clk), .D(D[2]), .Q(Q[2])),  FDQ3(.C(clk), .D(D[3]), .Q(Q[3])),  FDQ4(.C(clk), .D(D[4]), .Q(Q[4])),  FDQ5(.C(clk), .D(D[5]), .Q(Q[5])),  FDQ6(.C(clk), .D(D[6]), .Q(Q[6])),  FDQ7(.C(clk), .D(D[7]), .Q(Q[7]));    OR2 D0\_L(.I0(L\_0), .I1(R\_0), .O(D[0])),  D1\_L(.I0(L\_1), .I1(R\_1), .O(D[1])),  D2\_L(.I0(L\_2), .I1(R\_2), .O(D[2])),  D3\_L(.I0(L\_3), .I1(R\_3), .O(D[3])),  D4\_L(.I0(L\_4), .I1(R\_4), .O(D[4])),  D5\_L(.I0(L\_5), .I1(R\_5), .O(D[5])),  D6\_L(.I0(L\_6), .I1(R\_6), .O(D[6])),  D7\_L(.I0(L\_7), .I1(R\_7), .O(D[7]));    AND2 L0\_L(.I0(Q[1]), .I1(nS), .O(L\_0)),  L1\_L(.I0(Q[2]), .I1(nS), .O(L\_1)),  L2\_L(.I0(Q[3]), .I1(nS), .O(L\_2)),  L3\_L(.I0(Q[4]), .I1(nS), .O(L\_3)),  L4\_L(.I0(Q[5]), .I1(nS), .O(L\_4)),  L5\_L(.I0(Q[6]), .I1(nS), .O(L\_5)),  L6\_L(.I0(Q[7]), .I1(nS), .O(L\_6)),  L7\_L(.I0(s\_in), .I1(nS), .O(L\_7));    AND2 R0\_L(.I0(p\_in[0]), .I1(S), .O(R\_0)),  R1\_L(.I0(p\_in[1]), .I1(S), .O(R\_1)),  R2\_L(.I0(p\_in[2]), .I1(S), .O(R\_2)),  R3\_L(.I0(p\_in[3]), .I1(S), .O(R\_3)),  R4\_L(.I0(p\_in[4]), .I1(S), .O(R\_4)),  R5\_L(.I0(p\_in[5]), .I1(S), .O(R\_5)),  R6\_L(.I0(p\_in[6]), .I1(S), .O(R\_6)),  R7\_L(.I0(p\_in[7]), .I1(S), .O(R\_7));    INV nS\_L(.I(S), .O(nS));  endmodule |

1. 针对该模块，编写波形仿真输入代码
2. 进行波形仿真，并分析仿真结果是否正确

## 4.2 在xCal上增加串行、并行输入功能

1. 设计16位带并行输入的右移移位寄存器

|  |
| --- |
| `timescale 1ns / 1ps  module shift\_reg\_16(  input wire [2:0]btn,  input wire [3:0]switch,  input wire [3:0]p\_in,  output reg [15:0]op\_out1,  output reg [15:0]op\_out2  );  wire [0:15]Q1,Q2;  wire clk1 = btn[1]&switch[2]&~switch[3]&switch[0]&~switch[1];  wire clk2 = btn[1]&switch[2]&~switch[3]&switch[1]&~switch[0];    shift\_reg sr1(clk1, btn[2], btn[0], { 4{p\_in[0], p\_in[1], p\_in[2], p\_in[3]} }, Q1);  shift\_reg sr2(clk2, btn[2], btn[0], { 4{p\_in[0], p\_in[1], p\_in[2], p\_in[3]} }, Q2);    always @\* begin  if(switch == 4'b0101) op\_out1[15:0] = Q1[0:15];  else if(switch == 4'b0110) op\_out2[15:0] = Q2[0:15];  end    endmodule |

1. 增加到create\_operands模块中

|  |
| --- |
| `timescale 1ns / 1ps  module create\_oprands(  input wire [7:0] switch,  input wire [3:0] btn,  output reg [15:0] op\_out1,op\_out2  );  wire [15:0]out\_shift1, out\_shift2, out\_inc1, out\_inc2;    btn\_increment m0(btn[3:0], switch[1:0], out\_inc1, out\_inc2);  shift\_reg\_16 m1(btn[2:0], switch[3:0], switch[7:4], out\_shift1, out\_shift2);    always @\* begin  if(switch[1:0] == 2'b01)  case(switch[3:2])  2'b00:op\_out1 = out\_inc1;  2'b01:op\_out1 = out\_shift1;  endcase  else if(switch[1:0] == 2'b10)  case(switch[3:2])  2'b00:op\_out2 = out\_inc2;  2'b01:op\_out2 = out\_shift2;  endcase  end  endmodule  module btn\_increment(  input wire [3:0] btn,  input wire [1:0] switch,  output reg [15:0] op1,op2  );  initial op1 <= 16'b0110\_0000\_0000\_0000; //op1,op2初始化  initial op2 <= 16'b0110\_0000\_0000\_0000;    always @(posedge btn[0])begin  if(switch == 2'b01) op1[ 3: 0]<= op1[ 3: 0] + 1'd1;  else if(switch == 2'b10) op2[ 3: 0]<= op2[ 3: 0] + 1'd1;  end  always @(posedge btn[1])begin  if(switch == 2'b01) op1[ 7: 4]<= op1[ 7: 4] + 1'd1;  else if(switch == 2'b10) op2[ 7: 4]<= op2[ 7: 4] + 1'd1;  end  always @(posedge btn[2])begin  if(switch == 2'b01) op1[ 11: 8]<= op1[ 11: 8] + 1'd1;  else if(switch == 2'b10) op2[ 11: 8]<= op2[ 11: 8] + 1'd1;  end  always @(posedge btn[3])begin  if(switch == 2'b01) op1[ 15: 12]<= op1[ 15: 12] + 1'd1;  else if(switch == 2'b10) op2[ 15: 12]<= op2[ 15: 12] + 1'd1;  end    endmodule |

1. 修改相关代码

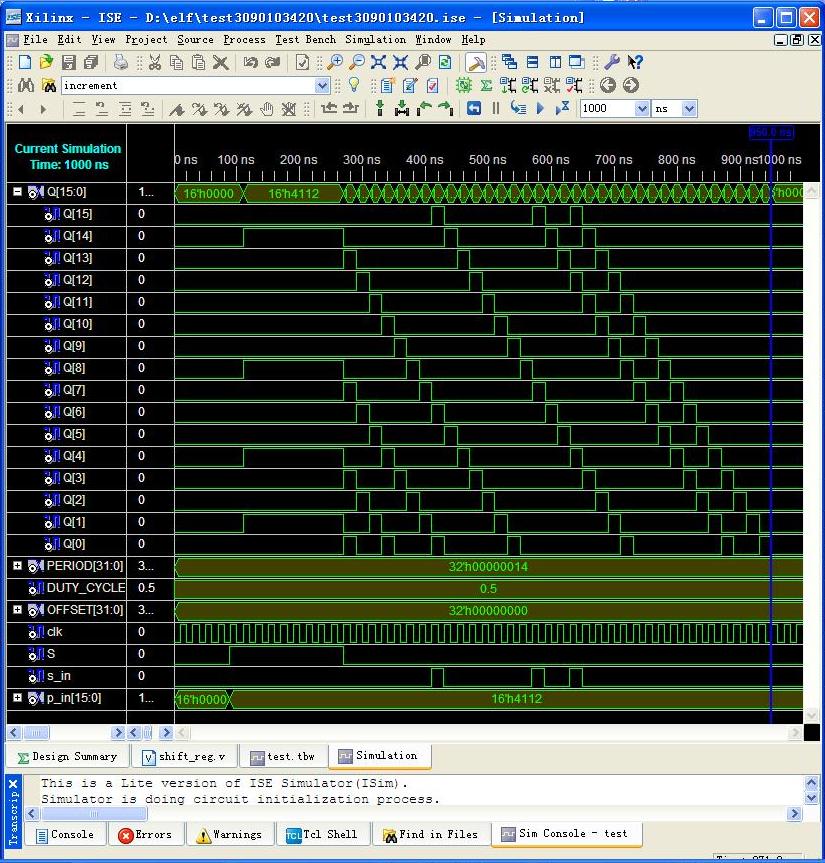
|  |
| --- |
| module top(  input wire clk,  input wire [3:0]btn\_in,  input wire[7:0]switch,  output wire [3:0]anode,  output wire [7:0]segment  );    //variable definition: op1, op2,disp\_num…  reg [15:0] display\_num;  wire [15:0] op1,op2,disp\_counter,disp\_clock;  wire [3:0] btn\_out;  wire [15:0] result;  wire clk\_1s;  pbdebounce b0(clk, btn\_in[0], btn\_out[0]);  pbdebounce b1(clk, btn\_in[1], btn\_out[1]);  pbdebounce b2(clk, btn\_in[2], btn\_out[2]);  pbdebounce b3(clk, btn\_in[3], btn\_out[3]);  display m0(clk, 0, display\_num, 4'b1111, anode, segment); //display module  calculate\_result m1(btn\_out[3:0], op1, op2, result); //calculate result  create\_oprands m2(switch[7:0], btn\_out[3:0], op1, op2); //generate op1/op2  counter\_1s m3(clk,clk\_1s);  counter\_16bit\_rev m4((clk\_1s&btn\_out[0]),switch[2],disp\_counter);  clock\_24h m5((clk\_1s & btn\_out[1]), disp\_clock);    always @\* begin  case (switch[1:0])  2'b01:begin //operand 1  display\_num = op1;  end  2'b10:begin //operand 2  display\_num = op2;  end  2'b00:begin //result  display\_num = result;  end  2'b11:begin //counter  if(btn\_out[1:0] == 2'b01)  display\_num = disp\_counter;  else if(btn\_out[1:0] == 2'b10)  display\_num = disp\_clock;  end  endcase  end  endmodule |

1. 修改ucf文件

|  |
| --- |
| NET "clk" LOC = "t9" ;  NET "btn\_in[0]" LOC = "m13" ;  NET "btn\_in[1]" LOC = "m14" ;  NET "btn\_in[2]" LOC = "l13" ;  NET "btn\_in[3]" LOC = "l14" ;  NET "switch[0]" LOC = "f12" ;  NET "switch[1]" LOC = "g12" ;  NET "switch[2]" LOC = "h14" ;  NET "switch[3]" LOC = "h13" ;  NET "switch[4]" LOC = "j14" ;  NET "switch[5]" LOC = "j13" ;  NET "switch[6]" LOC = "k14" ;  NET "switch[7]" LOC = "k13" ;  NET "segment[0]" LOC = "e14" ;  NET "segment[1]" LOC = "g13" ;  NET "segment[2]" LOC = "n15" ;  NET "segment[3]" LOC = "p15" ;  NET "segment[4]" LOC = "r16" ;  NET "segment[5]" LOC = "f13" ;  NET "segment[6]" LOC = "n16" ;  NET "segment[7]" LOC = "p16" ;  NET "anode[0]" LOC = "D14" ;  NET "anode[1]" LOC = "G14" ;  NET "anode[2]" LOC = "F14" ;  NET "anode[3]" LOC = "E13" ; |

1. 下载并验证

# 五、实验结果与分析



图表3 16位带并行输入的右移移位寄存器的调试仿真结果

* 16位带并行输入的右移移位寄存器的调试仿真结果正常

# 六、讨论、心得

本次实验增加了移位寄存器功能，较前几次实验需要修改、编写的代码更多了，功能模块也更复杂。但理解移位寄存器的原来，经过多次实验练习后，还是能够较顺利的完成实验任务。

经过这么多次的实验，我越来越发现Verilog编写逻辑电路的强大之处。它可以把复杂的逻辑电路拆分成多个逻辑关系紧密的模块，简单明了，再通过工业映射，实现规模生产。

我们在学习Verilog代码的同时，加深了对逻辑电路的原理及应用的理解。