**《数字逻辑》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | 陈鹏宇 | | **年级** | | 2020级 |
| **学号** | | 20204227 | | **专业、班级** | | 计算机科学与技术01班 |
| **实验名称** | 实验九 寄存器 | | | | | |
| **实验时间** | 2021/11/26 | | **实验地点** | | Ds1410 | |
| **实验成绩** |  | | **实验性质** | | **□验证性 □设计性 □综合性** | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  评语：  评价教师签名（电子签名）： | | | | | | |
| 一、实验目的  掌握寄存器的逻辑功能，并学会寄存器与寄存器组的设计与实现。 | | | | | | |
| 二、实验项目内容   1. 设计一个带有（同步或异步）复位的D触发器 ddf1。编写仿真文件观察仿真时序波形图，回答异步清零与同步清零的区别，并下载到板验证。 2. 给1）中的D触发器dff1增加一个load信号就成为了一个以为寄存器reg\_1，当load为1时，那么在下一个始终上升沿，输入值d就被存储到q中，编写代码实现reg\_1，并下载到板验证。 3. 把N个1位寄存器组合起来就构成了一个N位寄存器。只需要将1位寄存器的输入端和输出端定义为N位的数组。编写代码实现一个可变宽度的N位寄存器（默认为8），编写顶层模块实例化一个8位寄存器，并下载到板验证。 4. 实现一个8位的通用移位寄存器，可以加载并行数据，通过两位控制信号ctrl可以选择控制将其内容向左移位、向右移位或保持原有状态。利用此移位寄存器实现并转串（先加载并行输入，然后移位），和串转并（首先移位，然后进行并行输出），并下载到板验证。 5. 实现一个8位的桶形移位寄存器，具有8位数据输入和8位数据输出，以及指定如何移动数据的控制输入，指定移位方向、移位类型（循环、算术还是逻辑移位）及移动的位数（0~7）。信号和板上器件的关系是：原始数据d[7:0]--SW[7:0]，输出q[7:0]--ld[7:0]，s[2:0]--SW[15:13]表示移动的位数，t[1:0]--SW[12:11]表示移动类型（00逻辑右移，01-算术右移，1X-左移）。 | | | | | | |
| 三、实验设计  1.（1）同步时序         1. 异步时序          1. 波形图      1. 波形图     电路图      5. | | | | | | |
| 四、实验过程或算法(关键步骤、核心代码注解等）  1.源代码  module ddf1(  input wire clk,  input wire d,  input wire reset,  output reg q,  output reg qn  );  //同步时序电路  /\*  always @(posedge clk) begin  if(reset == 0) begin  q = 0 ; qn = 1;  end  else begin  q = d;  qn = ~d;  end  end  \*/  //异步时序电路  /\*\*/  always @(posedge clk or posedge reset) begin  if(reset == 0) begin  q <= 0 ; qn <= 1;  end  else begin  q = d;  qn = ~d;  end  end  endmodule  仿真：  module sim\_ddf1();  reg clk, reset, d;  wire q, qn;  always begin  #10 clk <= ~clk;  end  initial begin  clk = 0; d = 1; reset = 0;  #20 reset = 1;  #20 reset = 1; d = 0;  #20 reset = 0; d = 1;  #20 reset = 0; d = 0;  end  ddf1 uut(  .clk(clk),  .d(d),  .reset(reset),  .q(q),  .qn(qn)  );  Endmodule  2.源代码  module reg\_1(  input wire clk, input wire d, input wire reset, input wire load,  output reg q, output reg qn  );    always @(posedge clk or posedge reset or posedge load)  begin  if(reset == 0) begin  q = 0; qn = 1;  end  else begin  if(load == 1) begin  q = q; qn = ~q;  end  else begin  q = d; qn = ~d;  end  end  end  Endmodule  仿真代码：  module sim\_reg\_1();  reg clk, reset, load, d;  wire q, qn;  always begin  #10 clk = ~clk;  end  initial begin  clk = 0; reset = 1; load = 0; d = 0;  #20 d = 1; load = 1;  #20 d = 0;  #20 d = 1; load = 0;  #20 d = 0;  end  reg\_1 uut(  .clk(clk),  .d(d),  .reset(reset),  .load(load),  .q(q),  .qn(qn)  );  endmodule  3.源代码  module reg\_N(clk, reset, load, d, q);  parameter N = 8;  input wire clk, reset, load;  input wire [N-1:0] d;  output reg [N-1:0] q;  always @(posedge clk or posedge reset or posedge load)  begin  if(reset == 1) begin  q = 0;  end  else begin  if(load == 1) begin  q = q;  end  else begin  q = d;  end  end  end  Endmodule  仿真代码：  module sim\_reg\_N();  parameter N = 8;  reg clk, reset, load;  reg [N-1:0] d;  wire [N-1:0] q;  always begin  #10 clk = 0;  #10 clk = 1;  // clk = ~clk;  end  reg\_N uut(clk, reset, load, d, q);  initial begin  clk = 0; reset = 0; load = 0; d = 0;  #20 d = 1;  #20 d = 2;  #20 d = 4;  #20 d = 8; load = 1;  #20 d = 32;  #20 d = 16;  #20 d = 8; load = 0; reset = 1;  #20 d = 4;  #20 d = 2;  #20 d = 1;  end  Endmodule   1. 源代码（串并转化）   module shift\_reg(clk, reset, en, ctr, d, q);  input wire clk, reset, en;  input wire [1:0] ctr;  input wire d;  output [7:0] q;  reg [7:0] temp;  reg [31:0] selcnt = 31'b000; //时钟分频  reg clk100;  always @(posedge clk)  begin  if(selcnt == 100\_000\_000) //周期2s  begin  selcnt <= 0;  clk100 <= ~clk100;  end  else  selcnt <= selcnt + 1;  end  //串转并  always @(posedge clk100 or posedge reset or en)  begin  if(reset == 1) begin  temp = 8'b0;  end  else if(en == 1) begin  temp = {temp[6:0], d};  end  else begin  case(ctr)  0: temp = {temp[0] , temp[7:1]}; //循环右移  1: temp = {temp[6:0] , temp[7]}; //循环左移  2: temp = temp; //保持  3: temp = temp;  default temp = temp;  endcase  end  end  assign q = temp;  Endmodule  源代码（并串转化）  module shift\_reg(clk, reset, en, ctr, d, q);  input wire clk, reset, en;  input wire [1:0] ctr;  input wire [7:0] d;  output q;  reg [7:0] temp;  reg [31:0] selcnt = 31'b000; //时钟分频  reg clk100;  always @(posedge clk)  begin  if(selcnt == 100\_000\_000) //周期2s  begin  selcnt <= 0;  clk100 <= ~clk100;  end  else  selcnt <= selcnt + 1;  end  //并转串  always @(posedge clk100 or posedge reset or posedge en)  begin  if(reset == 1) begin  temp = 8'b0;  end  else if(en == 1) begin  temp = d;  end  else begin  case(ctr)  0: temp = {temp[0] , temp[7:1]}; //循环右移  1: temp = {temp[6:0] , temp[7]}; //循环左移  2: temp = temp; //保持  3: temp = temp;  default temp = temp;  endcase  end  end  assign q = temp[0];  Endmodule  5.源代码  module shift\_reg(clk, reset, en, d, ctr, cc, q);  input wire clk, reset, en;  input wire [7:0] d;  input wire [1:0] ctr; //控制移动类型  input wire [2:0] cc; //控制移位数  output [7:0] q;  reg [31:0] selcnt = 31'b000; //时钟分频  reg [15:0] cnt; reg[7:0] temp; reg clk100;  always @(posedge clk)  begin  if(selcnt == 100\_000\_000) //周期2s  begin  selcnt <= 0;  clk100 <= ~clk100;  end  else  selcnt <= selcnt + 1;  end    always @(posedge clk100 or posedge reset or posedge en)  begin  if(reset == 1) begin  temp <= 0; cnt <= 0;  end  else if(en) begin //使能赋值  temp <= d;  end  else begin //开始并行输出  cnt <= cnt + 1;  if(cnt < cc) begin  case(ctr)  0: temp <= {temp[0] , temp[7:1]};//循环右移  1: temp <= {1'b1 , temp[7:1]}; //算术右移  2: temp <= {temp[6:0] , temp[7]};//循环左移  3: temp <= {temp[6:0] , 1'b0}; //算术左移（逻辑左移）  default temp <= {temp[6:0] , temp[7]};  endcase  end  end  end  assign q = temp;  endmodule | | | | | | |
| 五、实验过程中遇到的问题及解决情况(主要问题及解决情况)  步骤1.源文件的实现没有问题，但编写仿真文件时，开始无法显示全部情况，后来发现每次时钟变化周期是20ns，但每次情况的改变是10ns，导致有一些情况在上升沿后改变，无法被仿真。  步骤2.在写条件语句时，需遍历所有情况，否则会报错  步骤3.N位寄存器在上板没有问题，但仿真一直跑不出来，简单添加一个finish语句后，发现clk信号一直没有生成，但是已经初始化了，    调试后发现应该是always clk = ~clk的问题，将其改为#10 clk = 0； #10 clk = 1；后仿真结果正常。  步骤4/5：一开始并未采用时钟分频，发现难以捕捉到led的变化，将100Mhz分频为0.5hz后，周期为2s，能观察到灯的流水变化。 | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程   C:/Users/111/AppData/Local/Temp/picturecompress_20211127011232/output_1.jpgoutput_1  IMG_20211127_011132  C:/Users/111/AppData/Local/Temp/picturecompress_20211127011907/output_1.jpgoutput_1  Sw1为d，sw0为复位，ld1为q，ld0为qn  当reset = 0时，无论d为多少，q = 0；  当reset = 1时，q = d；  上板显示正确。  **关于异步和同步时序的区别**，在1.中的波形图中，20ns时可以看出，d=1且不在触发沿时，同步时序电路，reset（低电平有效）信号由0变为1时，q会由0变为1，而异步时序电路中，q会保持不变。分析知，同步时序电路所有触发器的时钟信号连在一起，只有当时钟脉冲信号来的时候，状态才会改变，否则保持，具有稳定性。而异步时序电路所有触发器时钟信号并未连在一起，电路状态也能由外部输出直接引起，对于D触发器，set reset clk信号均会造成电路状态的改变。  2.  C:/Users/111/AppData/Local/Temp/picturecompress_20211127031158/output_4.jpgoutput_4C:/Users/111/AppData/Local/Temp/picturecompress_20211127031158/output_3.jpgoutput_3C:/Users/111/AppData/Local/Temp/picturecompress_20211127031158/output_2.jpgoutput_2C:/Users/111/AppData/Local/Temp/picturecompress_20211127031158/output_1.jpgoutput_1  Sw2为d，sw1为load（高电平有效），sw0为reset（低电平有效）。  开始时，d，load，reset都为高电平，q为1；此时将记忆q的状态，当load为1的时间段内，无论d如何变化，q都将保持；reset置低电平，q变为0。  3.  C:/Users/111/AppData/Local/Temp/picturecompress_20211127041054/output_4.jpgoutput_4C:/Users/111/AppData/Local/Temp/picturecompress_20211127041054/output_3.jpgoutput_3C:/Users/111/AppData/Local/Temp/picturecompress_20211127041054/output_2.jpgoutput_2C:/Users/111/AppData/Local/Temp/picturecompress_20211127041054/output_1.jpgoutput_1  Sw[7:0]为d，ld[7:0]为q，sw14为load（高电平），sw13为reset（高电平）。  初始load = reset = 0；reset置1后灯复位，load置1后，无论sw如何变化，ld都将保持当前状态。  4.该步骤是为了掌握并串和串并转换的差别，并行输入转为串行输出时，每输出一次就移位，而串行输入转为并行输出时，是先移位输入，再整体输出，体现在代码上有细微的差别。  5.sw[7:0]--d; sw[15:13]--移位数量;sw[12:11]--移位类型;sw[10:9]--使能和重置。  sw[12:11] = 0--循环右移，1--算术右移（最高位置1），2--循环左移，3--算术左移（逻辑）最低为置0；每次完成移位工作后，需手动重置，目的是将cnt重新置0，再开启使能，载入数据，关闭使能，开始移位。由于动态结果，图片无法说明问题，就没有贴图，经上板验证，实际结果与理论一致。  七、小组分工情况说明  陈鹏宇：自主完成实验，完成实验报告  帕肉合·帕尔哈提：自主完成实验 | | | | | | |