**《数字逻辑》实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **曹泽阳** | | **年级** | | **2022级** |
| **学号** | **20220764** | | **专业、班级** | | **计算机科学与技术 卓越01** |
| **实验名称** | **实验十五 摩尔状态机序列检测器** | | | | |
| **实验时间** | **2023.11.23** | **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）： | | | | | |
| 1. 实验目的   1.通过实验，掌握有限状态机的设计方法，并用摩尔状态机设计实现“1101”序列检测。 | | | | | | |
| 二、实验项目内容  1、设计“1101”序列检测的状态转换图。  2、并转串模块设计，利用移位寄存器原理，setd 端上升沿将 8 位输入数据  同时锁存到移位寄存器中,在后续的 8 个时钟周期内将 8 位数据逐次移位  输出。  3、调用并转串输出模块，使用 Verilog HDL 语言的行为描述方式实现一个  摩尔状态机，能检测一个 8 位的二进制数据中是否存在“1101”序列，如  果检测到该序列则指定的 LED 灯亮。  4、综合、实现、生成 bit 流，下载到开发板进行验证。 | | | | | | |
| 三、实验设计（实验原理、真值表、原理图等）    一共三个模块，debounce\_button，parallel\_serial，seq\_det\_moore。其中debounce\_button 是顶层模块，进行按键消抖并将按键的信号传入到parallel\_serial（并转串） 作为时钟信号，parallel\_serial 从并行输入中取出来的高位信号进入debounce\_button （1011检测模块），根据状态转换图进行判定。 | | | | | | |
| 1. 实验过程或算法   **按键消抖模块**  `timescale 1ns / 1ps  // Module Function:按键消抖  /\*  !按键一般是机械弹性开关，由于机械触点的弹性作用，机械触点断开、闭合时会伴随着一连串的抖动，  !这个抖动会使得按键输出的高低电平连续变化，而这并不是真正的按下按键，  !如果直接作为开关控制后续电路，就会造成电路的不稳定，因此，需要采用按键消抖。  !机械按键按下时候有一个不稳定的抖动期，这个时间大概在20ms以内，  !我们可以利用这个20ms的抖动期，当检测到按键电平变化时20ms计数器重新计数，  !若计数器达到20ms，证明按键电平变化以后的20ms内没有再发生电平变化，可以认为是按键真正被按下，  !将此时的按键状态放入寄存器进而控制后续电路。    \*/  module debounce\_button (clk,rst,key,key\_pulse,en, data\_i, data\_o,dout);  parameter       N  =  1;                      //要消除的按键的数量  input             clk;                        //!这里是真实的时钟信号  input             rst;                        //!复位信号  input   [N-1:0]   key;                        //输入的按键  output  [N-1:0]   key\_pulse;                  //按键动作产生的脉冲  input en;  input [7:0] data\_i;  output  data\_o,dout;  reg     [N-1:0]   key\_rst\_pre;                //定义一个寄存器型变量存储上一个触发时的按键值  reg     [N-1:0]   key\_rst;                    //定义一个寄存器变量储存储当前时刻触发的按键值  wire    [N-1:0]   key\_edge;                   //检测到按键由高到低变化是产生一个高脉冲  //利用非阻塞赋值特点，将两个时钟触发时按键状态存储在两个寄存器变量中  //!时钟用两个寄存器存储前一时刻和当前的key的值  always @(posedge clk  or  negedge rst)       //!时钟上升沿或者复位信号下降沿    begin      if (rst)        begin          key\_rst <= {N{1'b1}};                //初始化时给key\_rst赋值全为1，{}中表示N个1          key\_rst\_pre <= {N{1'b1}};        end      else        begin          key\_rst <= key;       //第一个时钟上升沿触发之后key的值赋给key\_rst,同时key\_rst的值赋给key\_rst\_pre          key\_rst\_pre <= key\_rst; //非阻塞赋值。相当于经过两个时钟触发，key\_rst存储的是当前时刻key的值，key\_rst\_pre存储的是前一个时钟的key的值        end    end  //!key\_edge一直用于记录key有没有变化  assign  key\_edge = key\_rst\_pre & (~key\_rst);//脉冲边沿检测。当key检测到下降沿时，key\_edge产生一个时钟周期的高电平  //!也就是说，如果前一时刻的值和此时的值是相反的，那就说明按钮变化了  reg [17:0]    cnt;                       //产生延时所用的计数器，系统时钟12MHz，要延时20ms左右时间，至少需要18位计数器  //产生20ms延时，当检测到key\_edge有效是计数器清零开始计数  always @(posedge clk or negedge rst)    begin      if(rst)              //!如果复位了，就全部置为0        cnt <= 18'h0;      else if(key\_edge)    //todo 如果是key\_edge此时为1，说明按键状态变化了，开始计时，cnt初始化为0，        cnt <= 18'h0;      else        cnt <= cnt + 1'h1; //! 说明前一时刻和当前的key的值没发生变化，那就计时到20ms    end  reg     [N-1:0]   key\_sec\_pre;                //延时后检测电平寄存器变量  reg     [N-1:0]   key\_sec;  //延时后检测key，如果按键状态变低产生一个时钟的高脉冲。如果按键状态是高的话说明按键无效  always @(posedge clk  or  negedge rst)    begin      if (rst)        key\_sec <= {N{1'b1}};      else if (cnt==18'h3ffff)            //! 说明保持状态20ms了，那么此时的key就是正确的按键状态        key\_sec <= key;    end  always @(posedge clk  or  negedge rst)    begin      if (rst)        key\_sec\_pre <= {N{1'b1}};      else        key\_sec\_pre <= key\_sec;    end  assign  key\_pulse = key\_sec\_pre & (~key\_sec);     //在产生一个时钟的高脉冲后，传入parallel\_serial的clk中，计数，往后读取并行的输入数据  parallel\_serial u1(                    .clk(key\_pulse), .reset(rst), .en(en), .data\_i(data\_i), .data\_o(data\_o),.dout(dout)/\*,.clk\_real()\*/                  );  //!这个模块是使用产生的脉冲模拟时钟  endmodule  **并转串模块**  `timescale 1ns / 1ps  module parallel\_serial(           clk, reset, en, data\_i, data\_o,dout         );  input clk, reset,en;  input [7:0] data\_i;  output  data\_o,dout;  //input clk\_real;  reg [7:0]  data\_buf;  localparam N = 3; //!对50Mhz的时钟进行分频(50MHZ/2^3)  //!localparam和parameter的区别是：  //!localparam是一种常量，它的值不能被修改，也不能被defparam语句或模块实例化时的参数赋值来覆盖。  //!localparam可以用于定义模块内部的常量，比如电路的配置或逻辑功能，  //!它可以保护常量的值不被意外或错误地改变。  //!parameter也是一种常量，但它的值可以被修改，可以被defparam语句或模块实例化时的参数赋值来覆盖。  //!parameter可以用于定义模块外部的常量，比如电路的参数或延迟时间，它可以实现模块的参数化或自定义。  reg [N-1:0] regN; //?根据regN的值决定取哪一位，利用溢出实现8个时钟周期一循环  always@(posedge clk, posedge reset)    begin      if(reset)                     //FPGA板上配有 5 个按键，当按键按下时，表示 FPGA 的相应输入脚为高电平。所以平时的状态为低电平，即按键是未按下的  “弹起”  状态。        regN <= 0;      else        regN <= regN + 1;//!在第一次按下按钮的时候，regN会变成1，所以下面的case一开始就是从3'b1开始的    end  always@(posedge clk or negedge reset)    if(reset)      data\_buf <= 8'b0;    else if(en)      /\*当en=0时，regN只能计数，不能执行以下传输代码，即不能改变data\_buf的值了,不能再往seq\_det\_moore中传入下一个data\_o了，      但是seq\_det\_moore还是会随着clk的进行，而继续检索1101序列，此时data\_o的值是之前传入的值，不变。      只有在en=1后的下一次posedge clk时，才可以继续传输值\*/      begin        case(regN[N-1:N-3])                  //!根据regN后三位轮流让data\_i中的每一位成为dat\_buf最高位          3'b0:            data\_buf <= data\_i[7:0];          3'b1:            data\_buf <= {data\_i[6:0],data\_i[7]};          //!{} 的操作顺序是从左到右，也就是说，data\_i[6:0] 的值会被放在 data\_buf 的低位，          //!data\_i[7] 的值会被放在 data\_buf 的高位。          /\*          在Verilog中，data\_i[6:0]表示的是data\_i的第0位到第6位。          这个位切片操作保持了原始的位顺序，也就是说，data\_i[6:0]的第0位是data\_i的第0位，          第6位是data\_i的第6位。          当我们在{data\_i[6:0],data\_i[7]}这个表达式中使用连接操作符{}时，          data\_i[6:0]的位顺序不会改变。也就是说，data\_i[6:0]的第0位会被放在data\_buf的低位，第6位会被放在更高的位。          例如，如果data\_i的值为8'babcdefgh（其中a是最高位，h是最低位），          那么data\_i[6:0]的值就是7'bcdefgh。在{data\_i[6:0],data\_i[7]}表达式中，          data\_i[7]（也就是a）会被放在最高位，          data\_i[6:0]（也就是bcdefgh）会被放在低位，所以data\_buf的值就会是8'abcdefgh。          !实际上应该从左往右升高          !也就是说          !{data\_i[5:0],data\_i[7:6]}  是：data\_i[5],4,3,,2,1,0,7,6          !这样data\_buf的高位就是data\_i[6]          \*/          3'd2:            data\_buf <= {data\_i[5:0],data\_i[7:6]};          3'd3:            data\_buf <= {data\_i[4:0],data\_i[7:5]};          3'd4:            data\_buf <= {data\_i[3:0],data\_i[7:4]};          3'd5:            data\_buf <= {data\_i[2:0],data\_i[7:3]};          3'd6:            data\_buf <= {data\_i[1:0],data\_i[7:2]};          3'd7:            data\_buf <= {data\_i[0],data\_i[7:1]};          default:            data\_buf <= data\_i[7:0];        endcase      end  assign data\_o = data\_buf[7];  seq\_det\_moore u1(                  .clk(clk),                  .reset(reset),                  .din(data\_o),//每按一次模拟 clk 的按键便传入一个开关的值                  .dout(dout)                );  endmodule  **检测模块**  `timescale 1ns / 1ps  module seq\_det\_moore(           input clk,           input reset,           input din,           output reg dout         );  //状态声明  localparam [2:0]             s0 = 3'b000,               //!给状态编号             s1 = 3'b001,             s2 = 3'b010,             s3 = 3'b011,             s4 = 3'b100;  /\*  !s\_0     =       3'b000;       第零状态为0  !S\_1     =       3'b001;       第一个状态是只有一个1  !S\_11    =       3'b010;       第二个状态是11  !S\_110   =       3'b011;       第三个状态是110  !S\_1101  =       3'b100;       第四个状态是1101  \*/  reg [2:0] current\_state,next\_state;            //!当前状态和下一个状态  always @(posedge clk, posedge reset)    begin      if(reset)            //FPGA板上配有 5 个按键，当按键按下时，表示 FPGA 的相应输入脚为高电平。所以平时的状态为低电平，即按键是未按下的  "弹起"  状态。        current\_state <= s0;      else        current\_state <= next\_state;    end  always @ \*    begin      case(current\_state)        s0:          if(din == 1'b1)            next\_state = s1;          else            next\_state = s0;        s1:          if(din == 1'b1)            next\_state = s2;          else            next\_state = s0;        s2:          if(din == 1'b0)            next\_state = s3;          else            next\_state = s2;        s3:          if(din == 1'b1)            next\_state = s4;          else            next\_state = s0;        s4:          if(din == 1'b1)            next\_state = s2;             //如果next\_state设为s1,是不重叠检测，设为s2是可重叠的          else            next\_state = s0;        default:          next\_state = s0;      endcase    end  always @\*    begin      if(next\_state == s4)        dout = 1;      else        dout = 0;    end  endmodule  **给出引脚绑定：**  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_i[7]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_i[6]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_i[5]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_i[4]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_i[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_i[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_i[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_i[0]}]  set\_property PACKAGE\_PIN W13 [get\_ports {data\_i[7]}]  set\_property PACKAGE\_PIN W14 [get\_ports {data\_i[6]}]  set\_property PACKAGE\_PIN V15 [get\_ports {data\_i[5]}]  set\_property PACKAGE\_PIN W15 [get\_ports {data\_i[4]}]  set\_property PACKAGE\_PIN W17 [get\_ports {data\_i[3]}]  set\_property PACKAGE\_PIN W16 [get\_ports {data\_i[2]}]  set\_property PACKAGE\_PIN V16 [get\_ports {data\_i[1]}]  set\_property PACKAGE\_PIN V17 [get\_ports {data\_i[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports clk]  set\_property IOSTANDARD LVCMOS33 [get\_ports data\_o]  set\_property IOSTANDARD LVCMOS33 [get\_ports dout]  set\_property IOSTANDARD LVCMOS33 [get\_ports en]  set\_property PACKAGE\_PIN R2 [get\_ports en]  set\_property PACKAGE\_PIN E19 [get\_ports data\_o]  set\_property PACKAGE\_PIN U16 [get\_ports dout]  set\_property IOSTANDARD LVCMOS33 [get\_ports {key[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {key\_pulse[0]}]  set\_property PACKAGE\_PIN W5 [get\_ports clk]  set\_property IOSTANDARD LVCMOS33 [get\_ports rst]  set\_property PACKAGE\_PIN U18 [get\_ports rst]  set\_property PACKAGE\_PIN T17 [get\_ports {key[0]}]  set\_property PACKAGE\_PIN L1 [get\_ports {key\_pulse[0]}]  **并转串仿真**  `timescale 1ns / 1ps  module parallel\_serial\_tb;  reg clk,reset,en;  wire data\_o;  wire dout;  reg [7:0] data\_i;  // Note: CLK must be defined as a reg when using this method  parameter PERIOD = 10;  always    begin      #(PERIOD/2)  clk = ~clk;    end  initial    begin      clk   = 0;      reset = 1'b1;      @(posedge clk)   reset = 1'b0;      en=1'b1;      data\_i = 8'b11011101;    end  parallel\_serial u1(                    .clk(clk), .reset(reset), .en(en), .data\_i(data\_i), .data\_o(data\_o) , .dout(dout)                  );  endmodule | | | | | | |
| 五、实验过程中遇到的问题及解决情况  一开始只实现了不重叠检测，当设置输入序列为11011011时只会检测出一个1011，后面将状态转换的状态四在输入1之后的状态设置为状态2即实现了重叠检测。上板之后发现结果与预期不符，第一个输入信号并没有显示出来，后面发现，是因为我的regN在第一次按下按钮时就会变成1，因此0位输入没有被输入进去，不过由于输入实现了循环，所以最终还是会检测出来。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  Y7}V_I6MDTMGYD(_6_~@HPY  电路图：  5V9BY652W]44GAL8T]4MSOX    当前输入为1时，一个灯亮，当前检测出1101，两个灯亮，如果当前输入为0，那么不亮。   1. 小组分工情况说明   曹泽阳负责模块设计和代码编写，潘兴烨和张斯宇负责调试分析与报告书写。 | | | | | | |