**湘潭大学实验报告**

课程名称：数字电路与逻辑电路 实验名称：序列信号检测器 实验日期：2022.5.28 班级：21级计算机科学与技术3班 学号：202105566416 姓名：熊心雨

1. **实验目的**

1、用状态机实现序列检测器的设计，了解一般状态机的设计与应用。  
2、熟悉Vivado，具备使用Verilog HDL编程、仿真并进行硬件测试的能力。

1. **实验内容及要求**

1、序列检测器设计   
通过 Verilog HDL语言编程，实现一个8位序列检测器，要求有1个串行数据输入端、1个时钟脉冲输入、1个异步清零端、1个8位预置码输入端、4位的状态码输出端和检测结果输出端，序列检测器模块图如图3-25所示。输入、输出端口设计具体要求如下：  
clr：异步清零信号，低电平有效；  
clk：序列检测器时钟脉冲输入，上升沿有效。  
ds：串行数据输入端   
setd：8位预置码输入端   
c：4位状态码输出端  
dc：检测结果输出 端  
2、选用开发工具和开发环境，用 HDL完成序列 图3-25 序列检测器模块图检测器状态机的设计编辑和仿真测试等步骤，给出仿  
真波形。  
3、提出2个及其以上的设计方案并进行比较，选择合适的实验方案。  
4、根据选用的方案，创建项目，进行功能仿真和编译下载，完成实验操作并做好数据记录。  
5、根据实验内容，写出实验报告，包括程序设计、软件编译、仿真分析、硬件测试和详细实验过程：给出程序分析报告、仿真波形图及其分析报告。

**三、实验设备与软件环境**

实验电脑设备规格：

设备名称：xd60510

处理器：Intel(R) Core(TM) i5-9500 CPU @ 3.00GHz 3.00GHz

系统类型：64位操作系统 基于x64的处理器

实验电脑Windows规格：

版本：windows 10专业版

操作系统版本：18362.329

实验箱设备：

设备名称：Cpu FPGA 教学实验系统V2.0

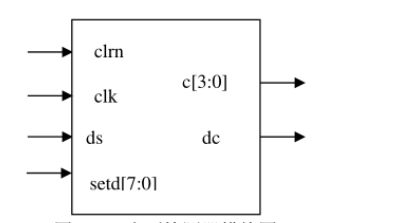
设备型号：LS-CPU-EXB-001

软件环境：

Vivado v2019.2 (64-bit)

1. **实验方案或原理**

序列检测器可用于检测一组或多组由二进制码组成的脉冲序列信号，当序列检测器连续接收到一组串行二进制码后，如果这组码与检测器中预先设置的码相同，则输出1，否则输出0。由于这种检测的关键在于正确码的收到必须是连续的，这就要求检测器必须记住前一次的正确码及正确序列，直到在连续的检测中所收到的每一位码都与预置数的对应码相同。在检测过程中，任何一位不相等都将回到初始状态重新开始检测。



序列检测器模块图  
 例如，若预先设置的二进制码为11010110，则当序列检测器连续接收到的 8位序列信号为11010110(时间顺序为自左向右)时，会输出1，表示检测到预置数据：否则输出0，表示没有检测到。

**五.代码设计及实验步骤**

1.代码设计

module sequencer(clk,clrn,din,ds);

input [7:0] din;

input clk,clrn;

output ds;

reg ds;

//3位二进制计数器

reg [2:0] count;

always @(posedge clk or negedge clrn)

begin

if ( !clrn )

begin

count <= 3'd0;

end

else

begin

count <= count +1;

end

end

//8选1数据选择器

always @( posedge clk )

case ( count )

3'd0 : ds <= din[7]; //7

3'd1 : ds <= din[6]; //6

3'd2 : ds <= din[5]; //5

3'd3 : ds <= din[4]; //4

3'd4 : ds <= din[3]; //3

3'd5 : ds <= din[2]; //2

3'd6 : ds <= din[1]; //1

3'd7 : ds <= din[0]; //0

default:ds <=1'b0;

endcase

endmodule

module sqdetector(ds,setd,clk,clrn,dc,c);

input ds,clk,clrn;

input [7:0] setd;

output [3:0] c;

output dc;

reg [3:0] c;

reg dc;

reg [3:0] n;

always @(c,setd,ds,n)

case (c)

4'b0000 : if(ds==setd[7]) begin n<=4'b0001; dc<=1'b0; end else begin n<=4'b0000; dc<=1'b0; end

4'b0001 : if(ds==setd[6]) begin n<=4'b0010; dc<=1'b0; end else begin n<=4'b0000; dc<=1'b0; end

4'b0010 : if(ds==setd[5]) begin n<=4'b0011; dc<=1'b0; end else begin n<=4'b0000; dc<=1'b0; end

4'b0011 : if(ds==setd[4]) begin n<=4'b0100; dc<=1'b0; end else begin n<=4'b0000; dc<=1'b0; end

4'b0100 : if(ds==setd[3]) begin n<=4'b0101; dc<=1'b0; end else begin n<=4'b0000; dc<=1'b0; end

4'b0101 : if(ds==setd[2]) begin n<=4'b0110; dc<=1'b0; end else begin n<=4'b0000; dc<=1'b0; end

4'b0110 : if(ds==setd[1]) begin n<=4'b0111; dc<=1'b0; end else begin n<=4'b0000; dc<=1'b0; end

4'b0111 : if(ds==setd[0]) begin n<=4'b0000; dc<=1'b1; end else begin n<=4'b0000; dc<=1'b0; end

default : begin n<=4'b0000; dc<=1'b0; end

endcase

always @(negedge clrn or posedge clk)

if (clrn==0) c<=0;

else c<=n;

Endmodule

module sqdetector\_display(

//时钟与复位信号

input clk,

input resetn, //后缀"n"代表低电平有效

//拨码开关，用于选择输入数

input input\_sel, //0:输入为序列检测器预置数据(setd)

//1:输入为序列信号发生器预置数据(din)

//脉冲开关，用于产生脉冲clock，实现单步执行

input clock,

//序列检测器数据检测结果输出端

output dc,

//序列检测器状态码输出端

output [3:0] c,

//触摸屏相关接口，不需要更改

output lcd\_rst,

output lcd\_cs,

output lcd\_rs,

output lcd\_wr,

output lcd\_rd,

inout[15:0] lcd\_data\_io,

output lcd\_bl\_ctr,

inout ct\_int,

inout ct\_sda,

output ct\_scl,

output ct\_rstn

);

//-----{调用sqdetector模块}begin

reg [7:0] setd; // sqdetector8位预置数输入端

wire ds;

sqdetector sqd(.ds(ds),.setd(setd),.clk(clock),.clrn(resetn),.dc(dc),.c(c));

//-----{调用sqdetector模块}end

//-----{调用sequencer模块}begin

reg [7:0] din; // sequencer预置数据输入端

sequencer sequencer\_module(.clk(clock),.clrn(resetn),.din(din),.ds(ds));

//-----{调用sequencer模块}end

//---------------------{调用触摸屏模块}begin--------------------//

//-----{实例化触摸屏}begin

//此小节不需要更改

reg display\_valid;

reg [39:0] display\_name;

reg [31:0] display\_value;

wire [5 :0] display\_number;

wire input\_valid;

wire [31:0] input\_value;

lcd\_module lcd\_module(

.clk (clk ), //10Mhz

.resetn (resetn ),

//调用触摸屏的接口

.display\_valid (display\_valid ),

.display\_name (display\_name ),

.display\_value (display\_value ),

.display\_number (display\_number),

.input\_valid (input\_valid ),

.input\_value (input\_value ),

//lcd触摸屏相关接口，不需要更改

.lcd\_rst (lcd\_rst ),

.lcd\_cs (lcd\_cs ),

.lcd\_rs (lcd\_rs ),

.lcd\_wr (lcd\_wr ),

.lcd\_rd (lcd\_rd ),

.lcd\_data\_io (lcd\_data\_io ),

.lcd\_bl\_ctr (lcd\_bl\_ctr ),

.ct\_int (ct\_int ),

.ct\_sda (ct\_sda ),

.ct\_scl (ct\_scl ),

.ct\_rstn (ct\_rstn )

);

//-----{实例化触摸屏}end

//-----{从触摸屏获取输入}begin

//根据实际需要输入的数修改此小节，

//建议对每一个数的输入，编写单独一个always块

//当input\_sel为0时，表示输入为序列检测器预置数据(setd)

always @(posedge clk)

begin

if (!resetn)

begin

setd <= 4'd0;

end

else if (input\_valid && input\_sel==1'b0)

begin

setd <= input\_value[7:0];

end

end

//当input\_sel为1时，表示输入数为序列信号发生器预置数据(din)

always @(posedge clk)

begin

if (!resetn)

begin

din <= 32'b0;

end

else if (input\_valid && input\_sel==1'b1)

begin

din <= input\_value[7:0];

end

end

//-----{从触摸屏获取输入}end

//-----{输出到触摸屏显示}begin

//根据需要显示的数修改此小节，

//触摸屏上共有44块显示区域，可显示44组32位数据

//44块显示区域从1开始编号，编号为1~44，

always @(posedge clk)

begin

case(display\_number)

6'd1 :

begin

display\_valid <= 1'b1;

display\_name <= "CLRN";

display\_value <= resetn;

end

6'd3 :

begin

display\_valid <= 1'b1;

display\_name <= "SETD";

display\_value <= {24'b0,setd};

end

6'd4 :

begin

display\_valid <= 1'b1;

display\_name <= "DIN";

display\_value <= {24'b0,din};

end

6'd5 :

begin

display\_valid <= 1'b1;

display\_name <= "C";

display\_value <= {28'b0,c};

end

6'd6 :

begin

display\_valid <= 1'b1;

display\_name <= "DC";

display\_value <= dc;

end

default :

begin

display\_valid <= 1'b0;

display\_name <= 40'd0;

display\_value <= 32'd0;

end

endcase

end

//-----{输出到触摸屏显示}end

//----------------------{调用触摸屏模块}end---------------------//

endmodule

`include "sqdetector.v"

module sqdetector\_tb;

reg clk,clrn,ds;

reg [7:0] setd;

wire [3:0] c;

wire dc;

sqdetector U(.ds(ds),.setd(setd),.clk(clk),.clrn(clrn),.dc(dc),.c(c));

initial

begin

ds=1;

clrn=0;clk=0; setd=8'b10101010;

#0 $display("time\tds\tsetd\tclk\tclrn\tdc\tc");

#4 clrn=1;

#40 setd=8'b01010101;

#40 setd=8'b00110011;

#40 setd=8'b11001100;

#40 setd=8'b11001101;

#40 setd=8'b10100101;

end

always #2 clk=~clk;

always #4 ds=~ds;

endmodule

#时钟信号

set\_property PACKAGE\_PIN AC19 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

#脉冲开关，用于输入作为复位信号，低电平有效

set\_property PACKAGE\_PIN Y3 [get\_ports resetn]

set\_property IOSTANDARD LVCMOS33 [get\_ports resetn]

#脉冲开关，用于输入作为单步执行的clock

set\_property PACKAGE\_PIN Y5 [get\_ports clock]

set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets clock\_IBUF]

set\_property IOSTANDARD LVCMOS33 [get\_ports clock]

#拨码开关的设置,用于输入sw0

set\_property PACKAGE\_PIN AC21 [get\_ports input\_sel]

set\_property IOSTANDARD LVCMOS33 [get\_ports input\_sel]

#LED灯1的设置,用于输出dc

set\_property PACKAGE\_PIN H7 [get\_ports dc]

set\_property IOSTANDARD LVCMOS33 [get\_ports dc]

#LED灯5~8的设置,用于输出c

set\_property PACKAGE\_PIN D5 [get\_ports {c[3]}]

set\_property PACKAGE\_PIN A3 [get\_ports {c[2]}]

set\_property PACKAGE\_PIN A5 [get\_ports {c[1]}]

set\_property PACKAGE\_PIN A4 [get\_ports {c[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {c[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {c[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {c[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {c[0]}]

#触摸屏引脚连接

set\_property PACKAGE\_PIN J25 [get\_ports lcd\_rst]

set\_property PACKAGE\_PIN H18 [get\_ports lcd\_cs]

set\_property PACKAGE\_PIN K16 [get\_ports lcd\_rs]

set\_property PACKAGE\_PIN L8 [get\_ports lcd\_wr]

set\_property PACKAGE\_PIN K8 [get\_ports lcd\_rd]

set\_property PACKAGE\_PIN J15 [get\_ports lcd\_bl\_ctr]

set\_property PACKAGE\_PIN H9 [get\_ports {lcd\_data\_io[0]}]

set\_property PACKAGE\_PIN K17 [get\_ports {lcd\_data\_io[1]}]

set\_property PACKAGE\_PIN J20 [get\_ports {lcd\_data\_io[2]}]

set\_property PACKAGE\_PIN M17 [get\_ports {lcd\_data\_io[3]}]

set\_property PACKAGE\_PIN L17 [get\_ports {lcd\_data\_io[4]}]

set\_property PACKAGE\_PIN L18 [get\_ports {lcd\_data\_io[5]}]

set\_property PACKAGE\_PIN L15 [get\_ports {lcd\_data\_io[6]}]

set\_property PACKAGE\_PIN M15 [get\_ports {lcd\_data\_io[7]}]

set\_property PACKAGE\_PIN M16 [get\_ports {lcd\_data\_io[8]}]

set\_property PACKAGE\_PIN L14 [get\_ports {lcd\_data\_io[9]}]

set\_property PACKAGE\_PIN M14 [get\_ports {lcd\_data\_io[10]}]

set\_property PACKAGE\_PIN F22 [get\_ports {lcd\_data\_io[11]}]

set\_property PACKAGE\_PIN G22 [get\_ports {lcd\_data\_io[12]}]

set\_property PACKAGE\_PIN G21 [get\_ports {lcd\_data\_io[13]}]

set\_property PACKAGE\_PIN H24 [get\_ports {lcd\_data\_io[14]}]

set\_property PACKAGE\_PIN J16 [get\_ports {lcd\_data\_io[15]}]

set\_property PACKAGE\_PIN L19 [get\_ports ct\_int]

set\_property PACKAGE\_PIN J24 [get\_ports ct\_sda]

set\_property PACKAGE\_PIN H21 [get\_ports ct\_scl]

set\_property PACKAGE\_PIN G24 [get\_ports ct\_rstn]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_rst]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_cs]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_rs]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_wr]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_rd]

set\_property IOSTANDARD LVCMOS33 [get\_ports lcd\_bl\_ctr]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[8]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[9]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[10]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[11]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[12]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[13]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[14]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {lcd\_data\_io[15]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_int]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_sda]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_scl]

set\_property IOSTANDARD LVCMOS33 [get\_ports ct\_rstn]

2.实验步骤

（1）创建工程

新建文件夹JG3，在该文件创建工程文件，并制定FPGA器件

1. 启动Vivado软件
2. 选择“RTL Project”，勾选“Do not specify sources at this time”
3. 指定FPGA器件
4. 出现总结界面时点击finnish，完成创建。

（2）模块设计

添加源文件。在其他文件编辑器里写好代码，添加到新建的工程中。

通过Verilog HDL语言编程，实现一个8位序列检测器，要求有1行串行输入端、1个时冲脉冲输入端、1个异步清零端、1个四位的状态输出端和检测结果输出端。

clr：异步清零信号，低电平有效；  
clk：序列检测器时钟脉冲输入，上升沿有效。  
ds：串行数据输入端   
setd：8位预置码输入端   
c：4位状态码输出端  
dc：检测结果输出 端

（3）功能仿真

a.建立一个测试模块，一个比较完备的测试模块能够产生输入激励信号，送入到要测试的功能模块中，然后读出该功能模块的执行结果，并与预期结果进行比较，以此来验证功能模块的正确性。

b.波形仿真

点击“Run Simulation”,选择“Run Behavioral Simulation”没有语法错误会弹出仿真波形界面。



**六．实验结果验证及分析**

预先设置的二进制码为11010110，则当序列检测器连续接收到的 8位序列信号为11010110(时间顺序为自左向右)时，会输出1，表示检测到预置数据：否则输出0，表示没有检测到。

实验结果表明与预设情况完全一致。

1. **实验过程中出现的问题及处理情况**

1. 问题： 时间设置不合理，波形图无法看到明显的现象。

处理办法：需重新设置仿真时钟信号，和清零端的时间间隔。

2.问题：可以显示波形图但是不能下载到实验箱上。

处理办法：把lcd\_module.dcp文件的代码绑定在vivado里面了。

**八.实验操作评分照片**

