# 实验题目：序列检测器设计

班级：无 42

学号：2014011058

姓名：陈誉博

日期：2016 年 4 月 23 日

# 一、 实验目的

1．掌握有限状态机的实现原理和方法；
2．掌握序列检测的方法。

# 二、 实验原理及设计思路

有限状态机是逻辑电路设计中经常要遇到的，在数字电路中，经常需要通过建立有限状态机的方式来进行时序数字逻辑的设计。在复杂数字系统设计中，有限状态机主要通过硬件描述语言视线，硬件描述语言能够清晰地描述状态转移过程和输入输出变量关系，使得时序逻辑设计大大简化，进而极大降低系统设计复杂度，提高系统模块化程度。

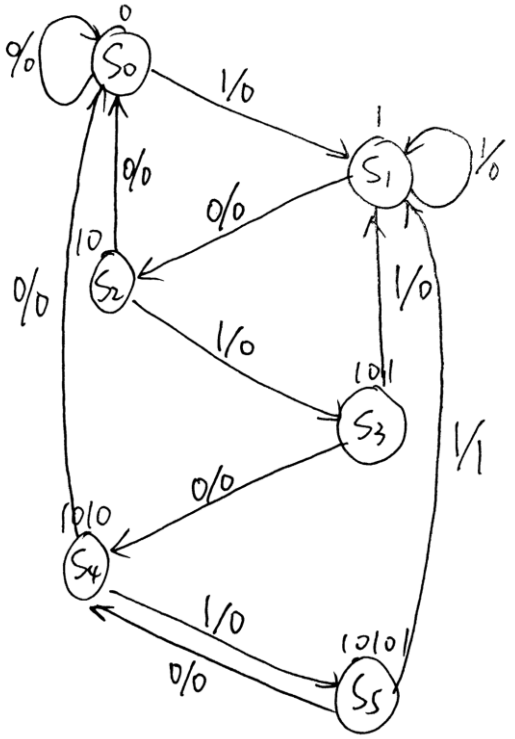
有限状态机从本质上来讲是由寄存器和组合逻辑构成的时序电路，各个状态之间的状态转移总是在时钟的触发下进行的。可以通过建立原始状态表和状态化简来设计电路。

本次实验要求用有限状态机和移位寄存器设计实现序列检测器。

设计思路：

1) 有限状态机：

状态转移图：




编码方式采用顺序编码:

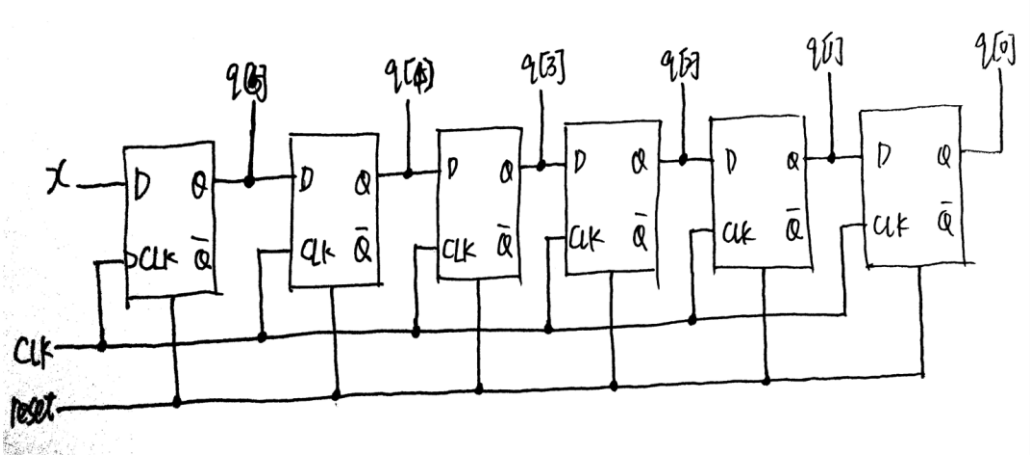
| 状态编号 | 状态编码 |
|---|---|
| $S_0$ | 000 |
| $S_1$ | 001 |
| $S_2$ | 010 |
| $S_3$ | 011 |
| $S_4$ | 100 |
| $S_5$ | 101 |

经卡诺图化简之后可以得出：

$$Q_2^+ = Q_2Q_0'x + Q_2Q_0x' + Q_1Q_0x'$$
$$Q_1^+ = Q_1Q_0'x + Q_2'Q_1'Q_0x$$
$$Q_0^+ = x$$
$$z = Q_2Q_0x$$

2)　移位寄存器：

电路图：




每当时钟上升沿来临，触发器的状态发生改变，移位寄存器的六个输出代表此时待检测序列的状态，由于时钟上升沿来临时只有一个数据输入，其他数据相应右移一位，符合实验要求中序列可以重复利用的条件。

## 三、 关键代码及源文件清单

1.　公共代码

1)　按键除抖代码 debounce.v:

```
module debounce(clk,key_i,key_o);
    input clk;
    input key_i;
    output key_o;

    parameter NUMBER = 24'd10_000_000;
    parameter NBITS = 24;

    reg [NBITS-1:0] count;
    reg key_o_temp;

    reg key_m;
        reg key_i_t1,key_i_t2;

    assign key_o = key_o_temp;

        always @ (posedge clk) begin
                key_i_t1 <= key_i;
                key_i_t2 <= key_i_t1;
        end

    always @ (posedge clk) begin
        if (key_m!=key_i_t2) begin
            key_m <= key_i_t2;
            count <= 0;
        end
        else if (count == NUMBER) begin
            key_o_temp <= key_m;
        end
        else count <= count+1;
    end
endmodule
```

2. 差异性代码

1) 有限状态机设计实现：zhuangtaiji.v

```
module zhuangtaiji(x,z,clk,anjian,reset,q);
input x,clk,reset,anjian;
output z;
output [2:0]q;
reg [2:0]q;
reg z;
wire chudou;
debounce db(.clk(clk),.key_i(anjian),.key_o(chudou));
always @(posedge reset or posedge chudou)
begin
        if(reset) begin
        z<=0;
        q<=0;
        end
        else begin
        q[2]<=(q[2]&&~q[0]&&x)||(q[2]&&q[0]&&~x)||(q[1]&&q[0]&&~x);
        q[1]<=(q[1]&&~q[0]&&x)||(~q[2]&&~q[1]&&q[0]&&~x);
        q[0]<=x;
        z<=q[2]&&q[0]&&x;
        end
end
endmodule
```

注：此处的代码实现用的是摩尔机，但是在设计电路的时候状态转移图画的是米利机。

有限状态机端口约束：constrains.xdc

```
set_property PACKAGE_PIN U16 [get_ports {z}]
set_property PACKAGE_PIN U15 [get_ports {q[0]}]
set_property PACKAGE_PIN U14 [get_ports {q[1]}]
set_property PACKAGE_PIN V14 [get_ports {q[2]}]
set_property PACKAGE_PIN T18 [get_ports {reset}]
set_property PACKAGE_PIN U17 [get_ports {anjian}]
set_property PACKAGE_PIN V16 [get_ports {x}]

set_property PACKAGE_PIN W5 [get_ports clk]
create_clock -period 10.000 -name CLK -waveform {0.000 5.000} [get_ports clk]
#set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports {z}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {reset}]
set_property IOSTANDARD LVCMOS33 [get_ports {anjian}]
set_property IOSTANDARD LVCMOS33 [get_ports {x}]
```

2) 移位寄存器设计实现:yiweijicunqi.v

```
module trigger(D,clk,reset,Q,nQ);
input D,clk,reset;
output Q,nQ;
reg Q,nQ;
always @(posedge reset or posedge clk)
begin
        if(reset) begin
        Q<=0;nQ<=1;
        end
        else begin
        Q<=D;nQ<=~D;
        end
end
endmodule
module jicunqi(x,reset,clk,anjian,q,z,last);
input x,reset,clk,anjian;
output [5:0]q;
output z;reg z;output last;reg last;
debounce db(.clk(clk),.key_i(anjian),.key_o(chudou));
trigger t1(.D(x),.clk(chudou),.reset(reset),.Q(q[0]),.nQ(n1));
trigger t2(.D(q[0]),.clk(chudou),.reset(reset),.Q(q[1]),.nQ(n2));
trigger t3(.D(q[1]),.clk(chudou),.reset(reset),.Q(q[2]),.nQ(n3));
trigger t4(.D(q[2]),.clk(chudou),.reset(reset),.Q(q[3]),.nQ(n4));
trigger t5(.D(q[3]),.clk(chudou),.reset(reset),.Q(q[4]),.nQ(n5));
trigger t6(.D(q[4]),.clk(chudou),.reset(reset),.Q(q[5]),.nQ(n6));
always @(posedge chudou or posedge reset)
begin
        if(reset) begin
        z<=0;last<=0;
        end
        else begin
        if((q==6'b010101||q==6'b110101)&&x==1) z<=1;
        else z<=0;
        last=x;
        end
end
endmodule
```

移位寄存器端口约束：constrains.xdc

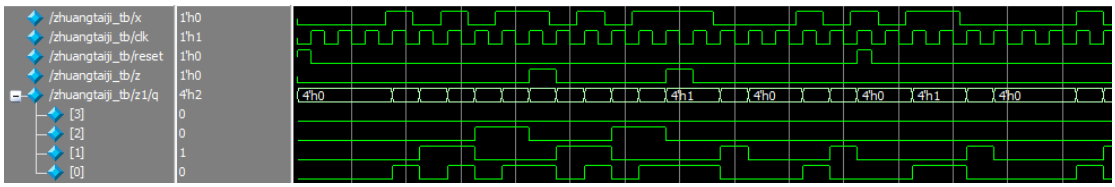
```
set_property PACKAGE_PIN U16 [get_ports {z}]
set_property PACKAGE_PIN E19 [get_ports {last}]
set_property PACKAGE_PIN U19 [get_ports {q[0]}]
set_property PACKAGE_PIN V19 [get_ports {q[1]}]
set_property PACKAGE_PIN W18 [get_ports {q[2]}]
set_property PACKAGE_PIN U15 [get_ports {q[3]}]
set_property PACKAGE_PIN U14 [get_ports {q[4]}]
set_property PACKAGE_PIN V14 [get_ports {q[5]}]
set_property PACKAGE_PIN T18 [get_ports {reset}]
set_property PACKAGE_PIN U17 [get_ports {anjian}]
set_property PACKAGE_PIN V16 [get_ports {x}]

set_property PACKAGE_PIN W5 [get_ports clk]
create_clock -period 10.000 -name CLK -waveform {0.000 5.000} [get_ports clk]
#set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports {z}]
set_property IOSTANDARD LVCMOS33 [get_ports {last}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {reset}]
set_property IOSTANDARD LVCMOS33 [get_ports {anjian}]
set_property IOSTANDARD LVCMOS33 [get_ports {x}]
```

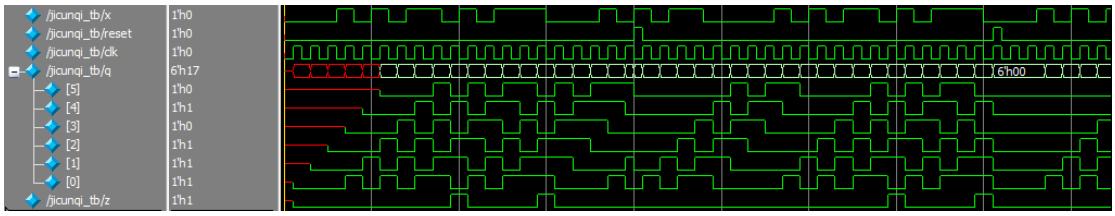
# 四、 仿真结果与分析

1. 有限状态机




最后四行为状态机当前状态的编码。

2. 移位寄存器



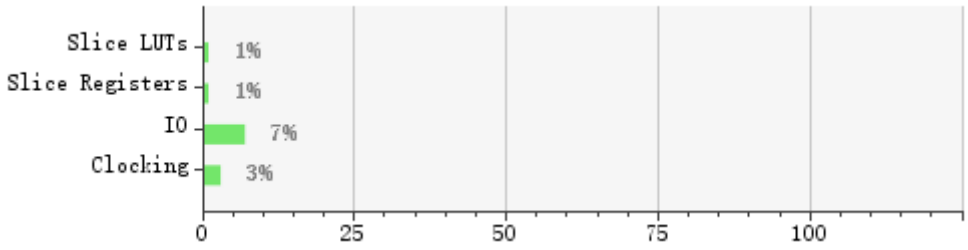

q 向量代表移位寄存器的数据。

# 五、 综合报告

1. 有限状态机

面积报告：

| Resource | Utilization | Available | Utilizat... |
|---|---|---|---|
| Slice LUTs | 34 | 20800 | 0.16 |
| Slice Registers | 32 | 41600 | 0.08 |
| IO | 8 | 120 | 6.67 |
| Clocking | 1 | 32 | 3.12 |



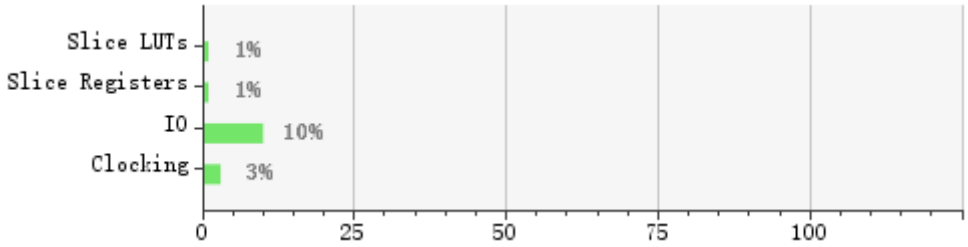

时序报告：

| Name | Waveform | Period (ns) | Frequency (MHz) |
|---|---|---|---|
| CLK | {0.000 5.000} | 10.000 | 100.000 |

2. 移位寄存器

面积报告：

| Resource | Utilization | Available | Utilizat... |
|---|---|---|---|
| Slice LUTs | 34 | 20800 | 0.16 |
| Slice Registers | 36 | 41600 | 0.09 |
| IO | 12 | 120 | 10.00 |
| Clocking | 1 | 32 | 3.12 |




时序报告：

| Name | Waveform | Period (ns) | Frequency (MHz) |
|---|---|---|---|
| CLK | {0.000 5.000} | 10.000 | 100.000 |

# 六、 硬件调试情况

1. 有限状态机

平台上 BTND 为时钟，BTNU 为 RESET，SW1 为输入，LD0 为输出显示，U15，U14 和 V14 为有限状态机当前状态的编码显示。输入 101011，每次输入时 U15,U14 和 V14 的状态变化为 000-001-010-011-100-101，当输入最后一个 1 时 U16 亮，显示序列检测正确。在任意时刻按下 RESET，所有灯全部变暗，序列归零，重新检测。

2. 移位寄存器

平台上 BTND 为时钟，BTNU 为 RESET，SW1 为输入，LD0 为输出显示，U19-V14 为移位寄存器的数据显示，E19 为输入数据显示。输入 10101101011，每次输入时 U19-V14 的状态变化为 000000-000001-000010-000101-001010-010101-101011-010110-101101-011010-110101-1101011，E19 的状态与输入完全相同，输入第 4 和第 7 个 1 时 U16 亮，显示序列检测正确。按下 RESET，所有灯全部变暗，序列归零，重新检测。

# 七、 实验总结与体会

此次实验的主要目的是为了熟悉有限状态机的硬件描述语言的实现方法。在编写代码之前我使用了在数字逻辑与处理器基础课程中学到的有限状态机的电路设计方法，所以最后的代码并没有体现出有限状态机的状态转移过程。助教告诉我最好采用直接的状态转移的方式编写，否则代码的重用性很差，如果被检测序列更改的话整个电路都需要重新设计，十分麻烦。还有本次实验中最大的收获就是更深刻的理解了米利机和摩尔机的区别。米利机的输出为组合逻辑电路，输出与输入直接相关，只要输入发生变化，不管时钟上升沿是否来临输出都会随之改变。摩尔机的输出为时序逻辑电路，只有在时钟上升沿来临的时刻输出才有可能发生变化。正是由于之前没有弄清楚这一点，在编写代码的时候把用米利机设计的电路实现成了摩尔机。多谢老师和助教的讲解让我及时明白这一点，帮助我更加深刻的理解了米利机和摩尔机的区别。