数字逻辑作业

习题 5

2019.11.3

数字逻辑作业—习题 5

一、 用 D 触发器设计可重叠 101 序列检测器，同时用 verilog 开发该模块，并开发测试程序

（一）熟悉同步时序逻辑电路的设计方法，使用 D 触发器设计 101 序列检测器

1.逻辑抽象

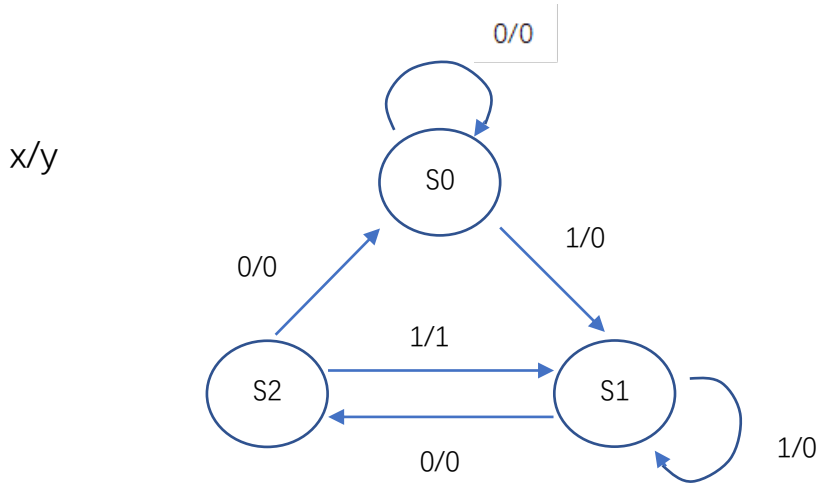
（1）分析给定的逻辑问题，确定输入变量、输出变量以及电路的状态数并定义输入、输出变量的含义，将电路状态顺序编号并将状态编码：

逻辑参量	符号及解释说明	
输入变量	x	序列检测器输入信号
输出变量	y	输出值（输出为 1 说明序列检测器检测到有效序列 101）
状态	S0	未接收到有效序列，编码为 00
	S1	接收到 1 序列或接收到 101 序列，编码为 01
	S2	接收到 10 序列，编码为 10

（说明：由于序列可重叠，故使用 S1 作为 1 序列和 101 序列公用状态）

（2）按照题意列出电路的原始状态转换表并画出 MEALY 型原始状态转换图：

现态	次态		输出 y	
	x=0	x=1	x=0	x=1
S0	S0	S1	0	0
S1	S2	S1	0	0
S2	S0	S1	0	1



2.求解电路方程

（1）作输入、现态、次态、激励关系表（二进制状态表）：

输入	现态$Q_1^nQ_0^n$	次态$Q_1^{n+1}Q_0^{n+1}$	激励T_1T_0
0	00	00	00
0	01	10	10
0	11	xx	xx
0	10	00	00
1	00	01	01
1	01	01	01
1	11	xx	xx
1	10	01	01

（2）求解输出函数和激励函数

分别作激励 T1、T0、输出 Z 的卡诺图并进行化简；

T1:

Q1Q0 X	00	01	11	10
0		1	x	
1			x	

T0:

Q1Q0 / X	00	01	11	10
0	1	1	x	1
1			x	

Z:

Q1Q0 / X	00	01	11	10
0			x	
1			x	1

得到： 输出函数：Z=XQ_1;

激励函数：T1=$\bar{X}Q_0$; T0=X

（二）使用 verilog 状态机开发该模块，并编写 testbench 测试程序

1.程序设计流程

参照 verilog 状态机设计流程，进行三段式 always 设计：

第一段 always 块采用同步时序逻辑，用于描述状态转移；

敏感变量为时钟上升沿或重置信号下降沿，块内行为描述为：若重置信号为低电平则将 S0 赋给现态，即重置状态；否则则让状态转移至次态。

第二段 always 块采用组合逻辑判断状态转移条件，描述状态转移规律；

敏感变量为现态或输入信号或重置信号，块内行为描述为：按照状态转移图进行设计，使用一个 case 语句，若现态为 S0，输入信号为 1 则将 S1 赋给次态；若现态为 S0，输入信号为 0，则将 S0 赋给次态；若现态为 S1，输入信号为 0，则将 S2 赋给次态；若现态为 S1，输入

信号为 1，则将 S1 赋给次态；若现态为 S2，输入信号为 1，则将 S1 赋给次态；若现态为 S2，输入信号为 0，则将 S0 赋给次态。按照如上逻辑编写代码。

第三段 always 块采用组合逻辑，描述状态机输出。

由于采用了 MEALY 型状态机，输出与输入信号直接相关，为了体现这一点性质，引入了临时输出变量 temp_z，用于记录现态得到的临时输出，再在最后一个 always 块中将其与输入信号进行比较，得到真正的输出信号 z。

```
1.  `timescale 1ns / 1ps
2.
3.
4.
5.  module FSM101(
6.      input clk,
7.      input rst,
8.      input a,
9.      output z
10.     );
11.
12.     reg z;
13.     reg temp_z;
14.     reg [1:0] currentstate,nextstate;
15.     parameter S0=2'b00;
16.     parameter S1=2'b01;
17.     parameter S2=2'b10;
18.
19.     always@(posedge clk or negedge rst)
20.         if(!rst)
21.             currentstate <= S0;
22.         else
23.             currentstate <= nextstate;
24.
25.     always@(currentstate or a or rst)
26.         if(!rst)
27.             nextstate=S0;
28.         else
29.             case(currentstate)
```

```
                S0: nextstate = (a==1)? S1:S0;
                S1: nextstate = (a==0)? S2:S1;
                S2: nextstate = (a==1)? S1:S0;
                default: nextstate=S0;
            endcase

    always@(rst or currentstate or a)
        if(!rst)
            temp_z=0;
        else
            case(currentstate)
                S0: temp_z = 0;
                S1: temp_z = 0;
                S2: temp_z = (a==1)?1:0;
                default: temp_z = 0;
            endcase

     always@(posedge clk or negedge rst)
        if(!rst)
            z <= 0;
        else
            begin
                if((temp_z == 1)&&(nextstate == S1))
                    z <= 1;
                else
                    z <= 0;
            end
endmodule
```

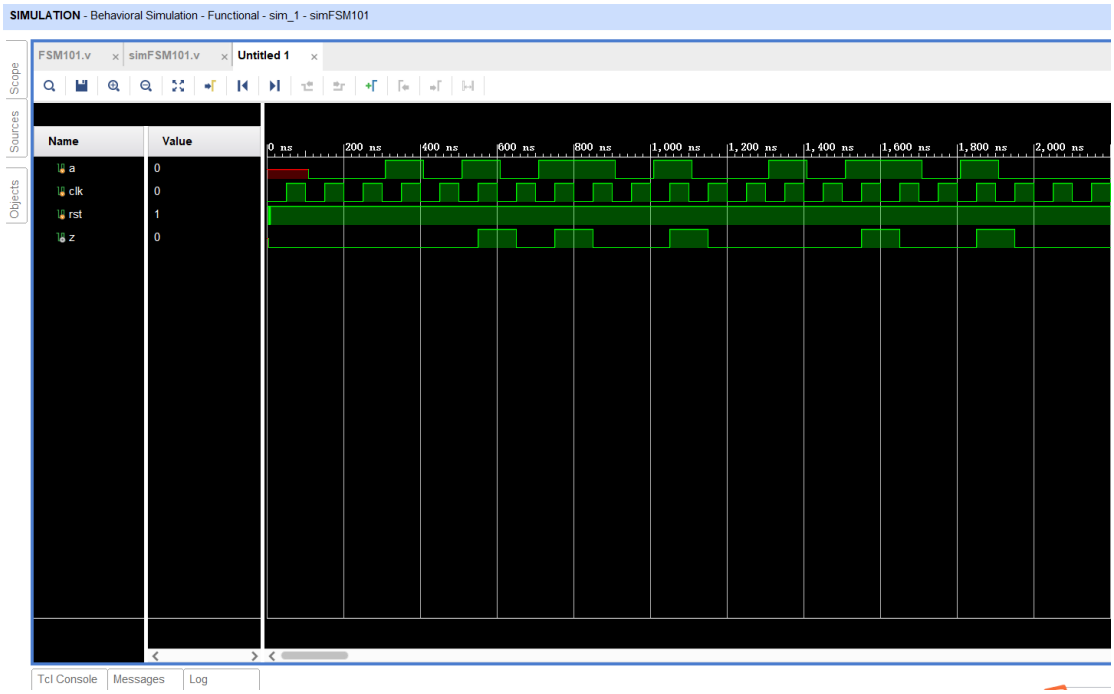
2.测试程序 testbench 编写

实例化 FSM101，并在 initial 块中改变输入信号的值，观察输出信号的变化情况，测试状态机序列选择器的功能。

```
`timescale 1ns / 1ps
module simFSM101(
    );
    reg a, clk, rst;
    wire z;
    FSM101 meal(.z(z),.clk(clk),.rst(rst),.a(a));
    initial begin
```

```
        clk = 0;
        rst = 1;
        #5 rst = 0;
        #3 rst = 1;
        #100 a = 0;
        #100 a = 0;
        #100 a = 1;
        #100 a = 0;
        #100 a = 1;
        #100 a = 0;
        #100 a = 1;
        #100 a = 1;
        #100 a = 0;
        #100 a = 1;
        #100 a = 0;
        #100 a = 0;
        #100 a = 1;
        #100 a = 0;
        #100 a = 1;
        #100 a = 1;
        #100 a = 0;
        #100 a = 1;
        #100 a = 0;
    end
    always #50 clk = ~clk;
endmodule
```

3.运行结果



从仿真波形中可以看出，

（1）当输入序列为 101 时，输出结果为 1；

（2）当输入序列重叠，只要序列出现 101 即可输出 1；

（3）未得到目标序列时，输出恒为 0。

从以上结论可知，设计的 101 序列选择器逻辑正确，功能正常，符合题目要求。

二、 设计通过识别输入的数据命令字 5306 来发动一次关键命令数据传输的数字系统，画出数字系统的状态图

（一）总体思路说明

基于题干要求和老师上课对该问题要求的分析，我个人对这个问题有两种不同的理解方式，这两种理解方式的共同点在于它们的关键环节都是使用时钟信号 CLK 进行计数操作，从而实现关键命令数据的有效生命时间延长为 100μs。区别在于：

理解方式（1）是把输入关键字 5306 当做并行输入，并且根据题干，仅仅把“5306”这一输入当做启动状态机后续功能的开关，系统接收到“5306”这一数据命令字之后，即发生状态转移，开始进行计数，在十个时钟周期之

内系统的输出都为1，关键命令数据传输系统（另一个状态机系统）接收到这个输出1之后即进行关键命令数据传输。

（注：理解方式（1）是指的仅仅把我们这次设计的这个数字系统当做一个“接收器”，“中转站”，即接收到我们想要的数据命令字之后，就输出信号1，让外部的其他数字系统接收到我们的输出信号1之后就启动它的功能。）

理解方式（2）是把总体输入当做一串数据字，当在这一串数据字串中检测到数据命令字段“5306”时即开始计数，把接下来100μs内接收到的数据字进行一次传输。

（例：比如在输入的数据字串为“530612sx5483/1s23012sd……”，系统检测到了数据命令字段“5306”，故开始计时100μs，在这100μs的时间内接收到的剩余数据字被当做关键命令数据传输出去。假设在100μs内输入到了3，则把字段“12sx5483”传输）

以上两种理解方式都可以作为一个工程中实用的数字系统进行设计，并且都符合本次练习题的题意。所以我将把这两种理解对应的数字系统均进行设计，画出它们的状态图。

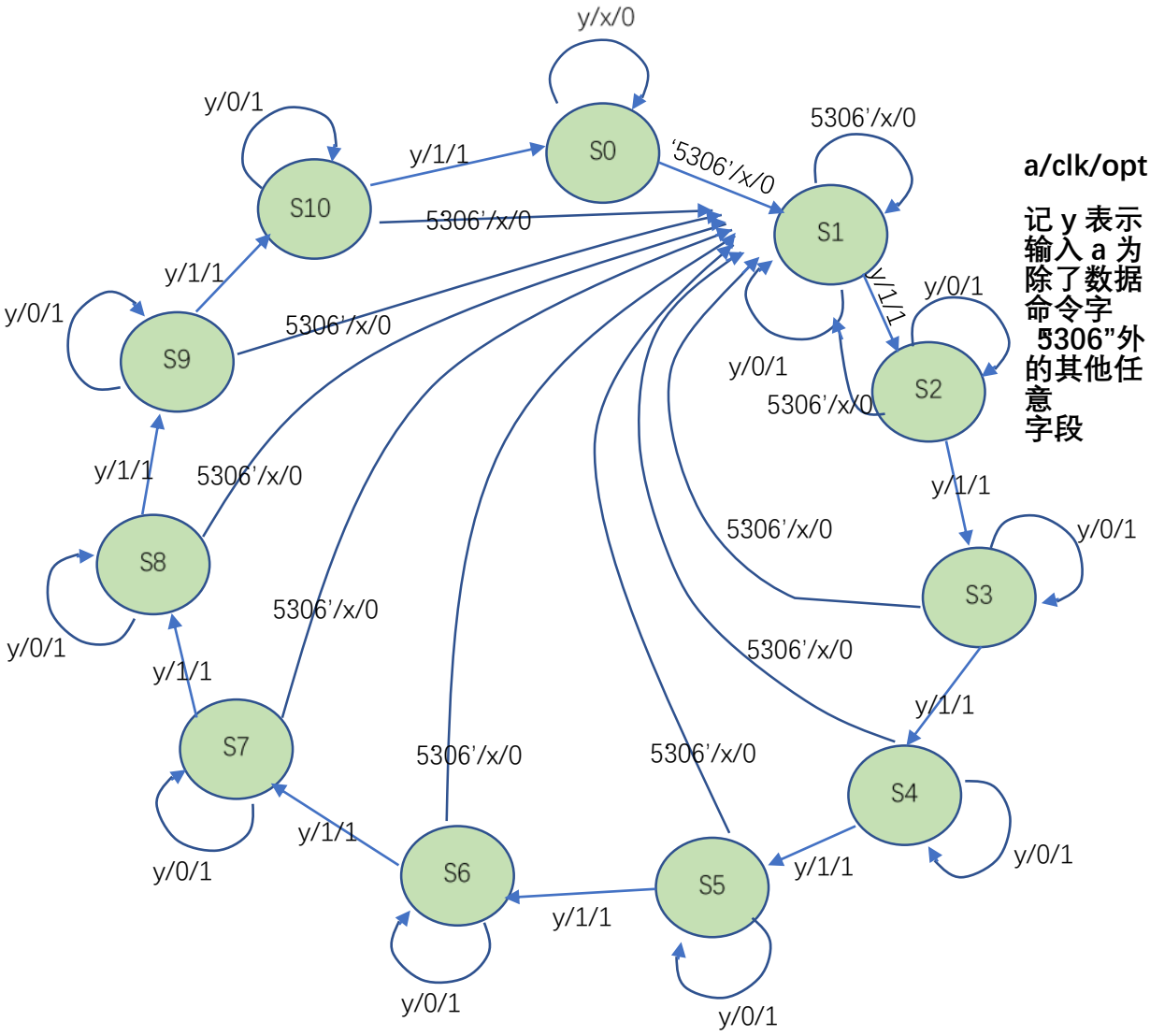
（二）理解方式1

1.逻辑抽象，状态设计与说明

逻辑参量	符号及解释说明	
输入变量	a	数字系统输入
	Clk	时钟信号（周期为10μs）
输出变量	Opt	输出值（输出为1说明系统接收到了数据命令字“5306”）
状态	S0	未接收到数据命令字“5306”
	S1	接收到数据命令字“5306”，开始计数
	S2	已计数1个时钟周期，即将开始计第2个时钟周期
	S3	已计数2个时钟周期，即将开始计第3个时钟周期
	S4	已计数3个时钟周期，即将开始计第4个时钟周期

	S5	已计数 4 个时钟周期，即将开始计第 5 个时钟周期
	S6	已计数 5 个时钟周期，即将开始计第 6 个时钟周期
	S7	已计数 6 个时钟周期，即将开始计第 7 个时钟周期
	S8	已计数 7 个时钟周期，即将开始计第 8 个时钟周期
	S9	已计数 8 个时钟周期，即将开始计第 9 个时钟周期
	S10	已计数 9 个时钟周期，即将开始计第 10 个时钟周期

2.根据状态转移关系，得到状态图

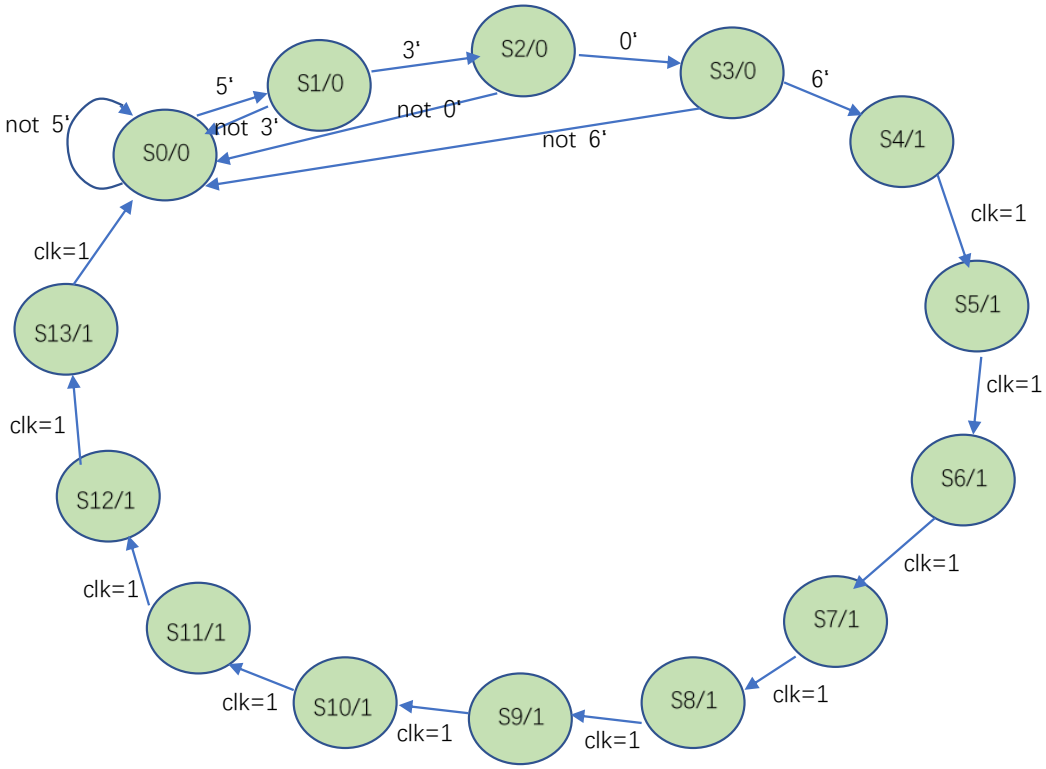


（二）理解方式 2

1.逻辑抽象，状态设计与说明

逻辑参量	符号及解释说明	
输入变量	a	数字系统输入
	Clk	时钟信号（周期为 10μs）
输出变量	Opt	输出值（输出为 1 说明系统接收到了数据命令字“5306”）
状态	S0	未接收到任何数据命令字
	S1	接收到数据命令字“5”
	S2	接收到数据命令字“53”
	S3	接收到数据命令字“530”
	S4	接收到数据命令字“5306”，即将开始计数
	S5	已计数 1 个时钟周期，即将开始计第 2 个时钟周期
	S6	已计数 2 个时钟周期，即将开始计第 3 个时钟周期
	S7	已计数 3 个时钟周期，即将开始计第 4 个时钟周期
	S8	已计数 4 个时钟周期，即将开始计第 5 个时钟周期
	S9	已计数 5 个时钟周期，即将开始计第 6 个时钟周期
	S10	已计数 6 个时钟周期，即将开始计第 7 个时钟周期
	S11	已计数 7 个时钟周期，即将开始计第 8 个时钟周期
	S12	已计数 8 个时钟周期，即将开始计第 9 个时钟周期
	S13	已计数 9 个时钟周期，即将开始计第 10 个时钟周期

2.根据状态转移关系，得到状态图



以上即本次设计的两种数字系统的状态图