上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

数字逻辑设计

LAB3 – 有限状态机

姓名：邱一航

学号：520030910155

1. 特定序列检测器

1.1 实验电路原理

序列检测器检测输入中的“101001”序列，检测到该序列则将输出置为 1，否则输出置为 0。该模块共有三个输入（时钟信号 clk、系统复位信号 rst_n、输入序列 din）和一个输出（输出检测信号 detector）。模块为上升沿触发，系统复位信号为异步复位、低电平有效。

我们使用 Moore 状态机来实现该功能。首先实现仅检测不重叠序列的序列检测器，如输入为 **101001**01001 仅在完成第六位输入后的下一个上升沿处（即第七个输入对应的上升沿）输出高电平；输入为 101001101001 才会在第六位后、第十二位后输出高电平。

不重叠序列检测器的状态转换表如下。state 括号中的内容解释了当前状态的意义，即当前已经检测到了括号内的序列。

States	Output	din=0	din=1
S0 (IDLE)	0	S0	S1
S1 (1)	0	S2	S1
S2 (10)	0	S0	S3
S3 (101)	0	S4	S1
S4 (1010)	0	S5	S3
S5 (10100)	0	S0	S6
S6 (101001)	1	S0	S1

表 1　不重叠序列检测器状态转换表

若需要检测重叠序列，如输入为 **10100**101001 时，需在第六位输入和第十一位输入处输出高电平，则状态转换表如下：

States	Output	din=0	din=1
S0 (IDLE)	0	S0	S1
S1 (1)	0	S2	S1
S2 (10)	0	S0	S3
S3 (101)	0	S4	S1
S4 (1010)	0	S5	S3
S5 (10100)	0	S0	S6
S6 (101001)	1	S2	S1

表 2　重叠序列检测器状态转换表

由于重叠序列检测器也能检测非重叠序列，因此我们这里直接实现进阶版本，即重叠序列检测器。重叠序列检测器共计有 7 个状态，其状态转移图如下。

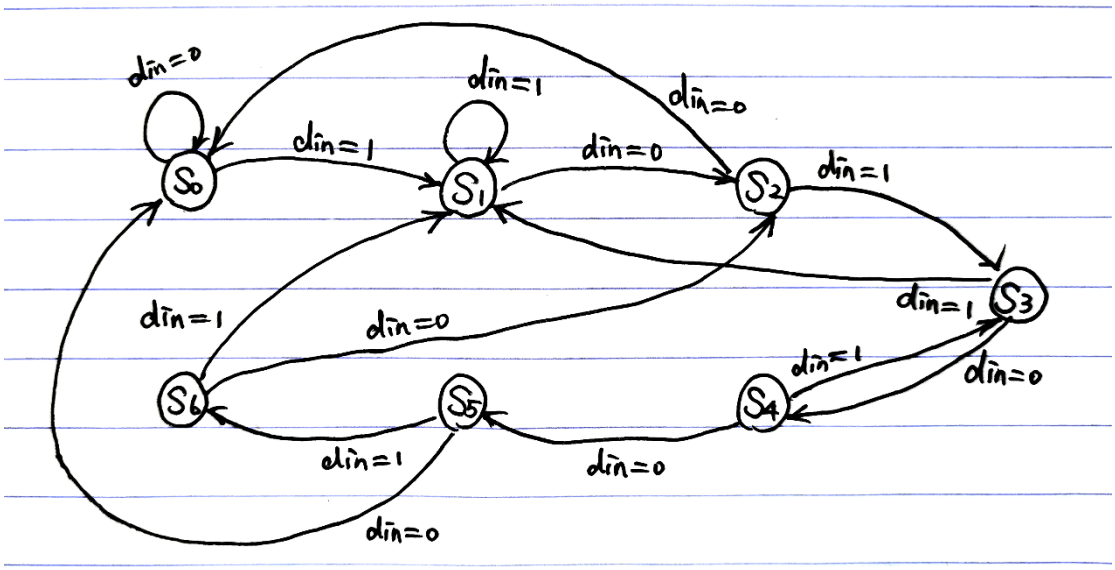


图 1　重叠序列检测器状态转移图

显然只有 S6 这一状态表示检测到了 101001 序列，此时输出为 1。记当前状态为 S，则输出表达式为(S==S6)。

1.2 电路实现与关键代码讨论

使用两段式状态机来实现序列检测器。状态转移表在 1.1 中已给出。笔者使用自然数计数的方式来编码七个不同的状态。

序列检测器的 Verilog 代码如下：

```
/* ================================[detector.v]================================ */
`timescale 1ns/100ps
module detector(
    input wire clk,
    input wire rst_n,
    input wire din,
    output reg detector
);

reg[2:0] state, next_state;

parameter S0 = 0;  // IDLE
parameter S1 = 1;  // 1
parameter S2 = 2;  // 10
parameter S3 = 3;  // 101
parameter S4 = 4;  // 1010
parameter S5 = 5;  // 10100
parameter S6 = 6;  // 101001

always @(posedge clk or negedge rst_n)
begin
    if (!rst_n)
    begin
        state <= S0;   // reset
        next_state <= S0;
    end
    else
    begin
        state <= next_state;
    end
end
```

```
always @(posedge clk)    //状态转移
begin
    case ({next_state,din})
        4'b0000:   next_state = S0;
        4'b0001:   next_state = S1;
        4'b0010:   next_state = S2;
        4'b0011:   next_state = S1;
        4'b0100:   next_state = S0;
        4'b0101:   next_state = S3;
        4'b0110:   next_state = S4;
        4'b0111:   next_state = S1;
        4'b1000:   next_state = S5;
        4'b1001:   next_state = S3;
        4'b1010:   next_state = S0;
        4'b1011:   next_state = S6;
        4'b1100:   next_state = S2;
        4'b1101:   next_state = S1;
        default:   next_state = S0;
    endcase
end

assign detector = (state==S6);

endmodule
```

```
/* ============================END of [detector.v]============================ */
```

在重复序列检测器的 testbench 中，我们首先枚举了所有可能的连续 12 位输入进行测试。此外，也可以通过随机的方式生成输入。

测试平台的 Verilog 代码如下：

```
/* ============================[detector_tb.v]================================ */

`timescale 1ns/100ps
module detector_tb;
reg CLK;
reg RST;
reg IN;
wire OUT;
reg[5:0] SEQ;     // in inverse order
reg[5:0] NEXT;    // in inverse order
reg[11:0] CUR;
reg[3:0] count_rst;
reg[3:0] count;
detector DTR(
    .clk(CLK),
    .rst_n(RST),
    .din(IN),
    .detector(OUT)
);

initial fork
    CLK = 1;
    RST = 0;
    IN = 1;
    count_rst = 0;
    SEQ = 6'b100101;
        // in inverse order
    NEXT = 6'b100101;
        // in inverse order
    CUR = {NEXT,SEQ};
    count = 12;
join

always #11  //测试 RST 是否异步
begin
    if (count_rst>11)
    begin
        RST = 1;
    end
    else
    begin
        if (count_rst<11)
            RST = 1;
        else
            RST = 0;
        count_rst = count_rst+1;
    end
end
```

```
    always #5
    begin
        CLK = ~CLK;
    end
    always #10
    begin
        IN = CUR % 2;
        CUR = CUR >> 1;
        count = count-1;
        if (count==0)
        begin
            NEXT = NEXT + 1;
            if (NEXT==0)
                SEQ = SEQ + 1;
            CUR = {NEXT,SEQ};
// 实际上可以直接生成 12 位 CUR，使用 NEXT 和
SEQ 主要是为了方便仿真波形时进行调试。
            count = 12;
        end
    end

endmodule
```

```
/* ============================END of [detector_tb.v]============================ */
```

使用随机方式生成输入的 testbench 代码如下。

```
/* ============================[detector_tb_rand.v]============================ */
`timescale 1ns/100ps
module detector_tb_rand;
reg CLK;
reg RST;
reg IN;
wire OUT;
detector DTR(
    .clk(CLK),
    .rst_n(RST),
    .din(IN),
    .detector(OUT)
);
initial fork
    CLK = 1;
    RST = 0;
    IN = 1;
join

always #5
begin
    CLK = ~CLK;
    RST = 1;
end

always #10
begin
    IN = {$random}%2;
end

endmodule
```

```
/* ============================END of [detector_tb_rand.v]======================== */
```

特别地，我们单独测试了持续输入首尾重叠的 101001 序列的情况，测试平台代码如下。

```
/* ============================[detector_tb2.v]============================== */
`timescale 1ns/100ps
module detector_tb2;
reg CLK;
reg RST;
reg IN;
wire OUT;
reg[3:0] count;
detector DTR(
    .clk(CLK),
    .rst_n(RST),
    .din(IN),
    .detector(OUT)
);
```

```
initial fork
    CLK = 1;    RST = 0;
    IN = 1;    count = 0;
join

always #5
begin
    CLK = ~CLK;
    RST = 1;
end

always #10
begin
    case (count)
        0: IN = 0;
        1: IN = 1;
        2: IN = 0;
        3: IN = 0;
        default: IN = 1;   //4
    endcase
    count = count + 1;
    if (count>4)
        count = 0;
end

endmodule

/* ============================END of [detector_tb2.v]========================== */
```

1.3 仿真波形结果讨论

重叠序列检测器程序在测试平台上的输出仿真波形如下。使用软件为 ModelSim 2020.3。

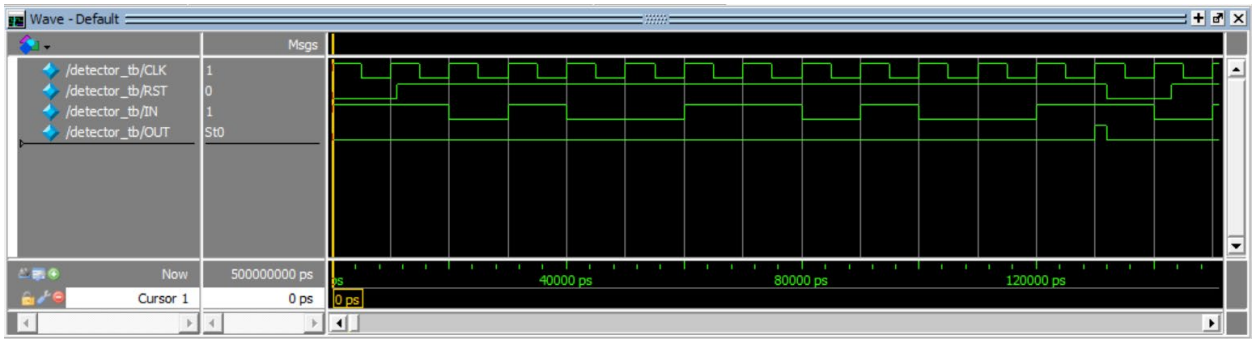


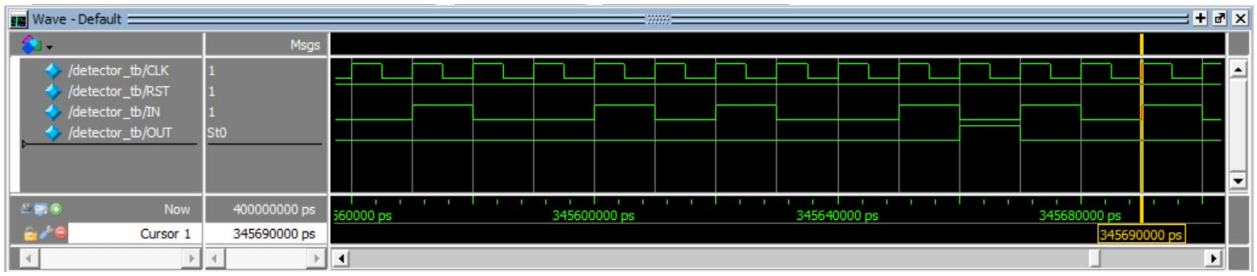
图 1　重叠序列检测器仿真波形

首先在 detector_tb 这一 testbench 上测试重叠序列检测器的异步置位信号工作情况。在 70ns~130ns 区段输入信号出现了“101001”序列，因此模块在 130ns 处产生高电平输出，但在 132ns 时 rst_n 置位端的输入由高电平变为低电平，132ns 处的输出随之变为低电平。由此可见，模块的 rst_n 置位段确实是异步的。

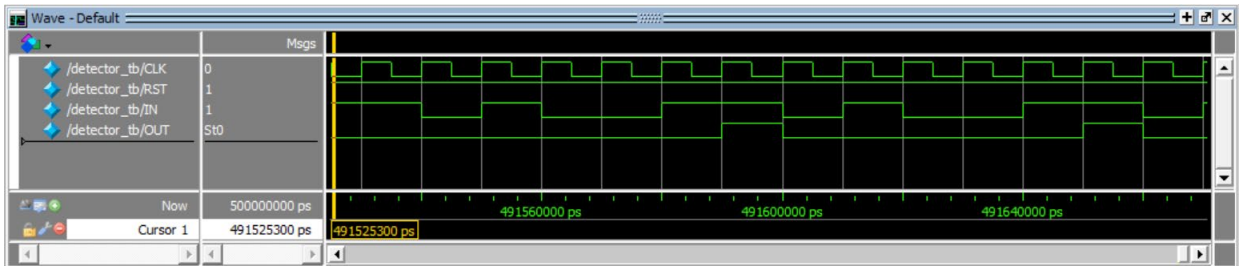
再对模块功能进行测试。由于 detector_tb 测试平台枚举了连续 12 位的可能输入，因此要枚举完毕所有可能情况（$2^6 \times 2^6 = 4096$种），至少需要模拟运行$(4096 \times 12 + 12) \times 10 = 491640ns$。（完成每种情况的输入需要 12 个时钟周期，最开始 12 个时钟周期用于验证 rst_n 是否是异步置位端，每个时钟周期为 10ns。）

在实验中，笔者模拟运行了 500μs。

由图 2(a)（见下页）可见，345600ns~345660ns 区段的输入信号为 101001，模块在 345670ns 时钟上升沿处产生了高电平输出，符合要求。



(a)



(b)

图 2　重叠序列检测器仿真波形

由图 2(b)可见，491530ns~491650ns 的输入序列为“1010011010011”，模块在两个序列结束后的第一个时钟上升沿（即 491590ns 处和 491650ns 处）产生了高电平输出，符合要求。

再使用 detector_tb2 对连续输入首尾重叠 101001 序列的情况进行测试。仿真波形如下。

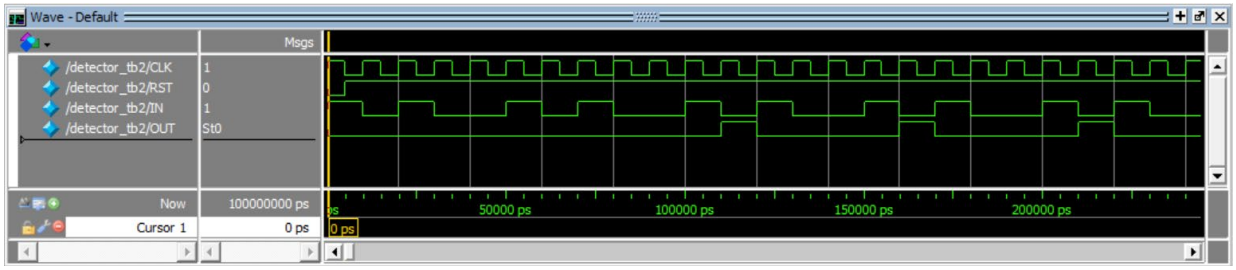


图 3　重叠序列检测器仿真波形

输入序列在 0ns~60ns 完成了“101001”的输入，但是 rst_n 端收到的信号从 5ns 开始才变为高电平，因此模块在 60ns 处没有高电平输出体现了复位端的正常工作。

50ns~210ns 的输入为“1010010100101001”，在三个首尾重叠的“101001”序列结束后的第一个时钟上升沿（即 110ns、160ns、210ns 处）均有高电平输出。因此该模块能判断重叠的 101001 序列。

2. 实验中的问题与建议

发现如果 A 是超过 1 位的 reg 类型时，对其使用左移有的时候会出现问题（A<<6）。因此实验中最后使用了拼接运算符，即{NEXT,SEQ}。