数字逻辑与处理器基础实验
第二次作业

无 81 马啸阳 2018011054

2020 年 4 月 11 日

1 实验目的

- 掌握时序电路设计方法
- 掌握 Verilog 设计进阶和仿真
  - 设计一个具有异步复位控制的 4bits 十进制同步加法计数器
  - 用有限状态机设计序列检测器，掌握有限状态机的设计方法
  - 用移位寄存器和组合逻辑实现序列检测器

2 实验原理

2.1 加法计数器

加法计数器框图如图 1所示，异步加法计数器单元输入复位与时钟信号，在时钟上升沿加一，输出四位 BCD 码计数值并送入 BCD 译码器，输出到七段数码管显示，同时七段数码管低电平使能。本实验中，加法计数器使用行为级设计方法实现。

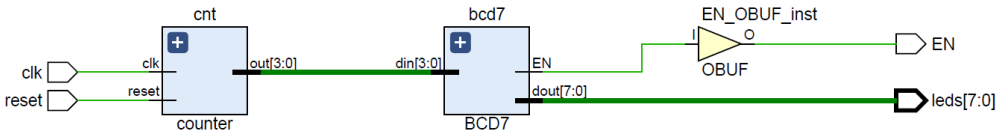

图 1: 加法计数器

2.2 有限状态机序列检测器

有限状态机框图如图 2所示，采用了 Mealy 机的设计，具有一个异步复位信号，状态存储于图中间的 D 触发器中，而左侧为根据输入与现态，通过多路选择器计算次态，作为触发器输入。输出由状态和输入共同经组合逻辑决定（经过一个触发器避免对输入的异步变化）。

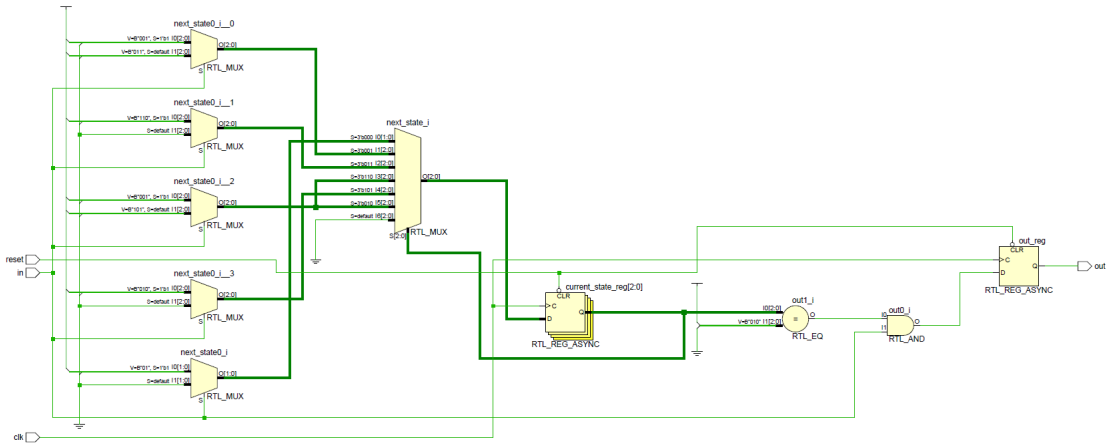

图 2: 有限状态机序列检测器

用于检测序列 101011 的有限状态机状态转移图如图 3所示。

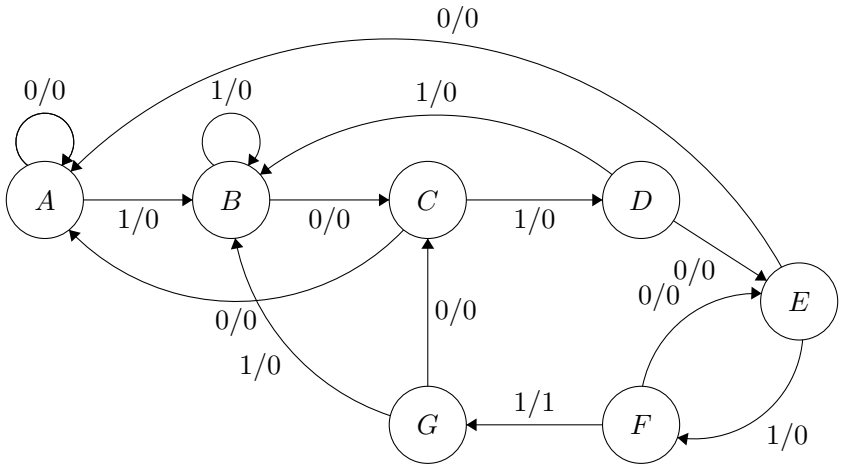

图 3: 有限状态机状态转移表

其中状态 A、B、C、D、E、F、G 分别表示输入序列（截至此时刻的最长匹配）为空、1、

10、101、1010、10101、101011。利用蕴含表法简化状态，合并 B、G，剩余六个状态，次态与输出表如表 1。

现态	次态		输出	
	输入 1	输入 0	输入 1	输入 0
*A	B	A	0	0
B	B	C	0	0
C	D	A	0	0
D	B	E	0	0
E	F	A	0	0
F	B	E	1	0

表 1: 有限状态机次态输出表

根据有限状态机的状态编码规则可给状态赋如表 2编码：

Q_2 \ Q_1Q_0	00	01	11	10
0	A	B	C	F
1		E		D

表 2: 有限状态机状态赋值

但实际上在综合后，FPGA 中有限状态机会自动改为顺序编码（因为使用查找表时，根据编码规则编码节省组合逻辑没有意义），也可以使用独热码等其它编码方法。

2.3 移位寄存器序列检测器

移位寄存器实现的序列检测器框图如图 4所示。检测“101011”这个六位序列时，移位寄存器存储最近收到的六个输入信号，在时钟上升沿触发移位操作，存入最新的串行输入值，丢弃 6 个时钟周期前的输入。而序列检测只需将移位寄存器中所存的数据与待检测序列进行比较。使用行为级设计方法时，不必用 D 触发器实现移位寄存器，只需使用 Verilog 的拼接运算符即可。

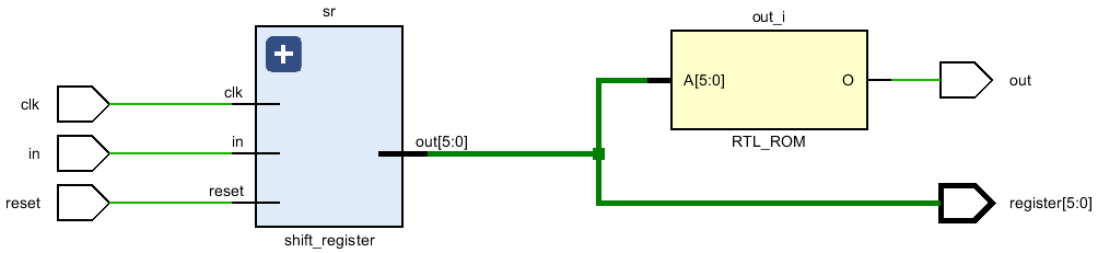

图 4: 移位寄存器序列检测器

3 实验代码

3.1 文件清单

```
.
├── counter ..... 加法计数器
│   ├── BCD7.v ..... 七段译码管模块
│   ├── counter.v ..... 加法计数器模块
│   ├── top.v ..... 顶层模块
│   ├── top_tb.v ..... testbench
│   └── top.xdc ..... 管脚约束
└── sequence_detector
    ├── fsm ..... 有限状态机序列检测器
    │   ├── fsm.v ..... 有限状态机
    │   ├── fsm_tb.v ..... testbench
    │   └── fsm.xdc ..... 管脚约束
    └── shift_register ..... 移位寄存器序列检测器
        ├── shift_register.v ..... 移位寄存器
        ├── detector.v ..... 序列检测器
        ├── detector_tb.v ..... testbench
        └── detector.xdc ..... 管脚约束
```

3.2 加法计数器

带异步复位的 4 位 BCD 码加法计数器代码如下:

```
module counter (reset, clk, out);
input reset;
input clk;
output [3:0] out;

reg [3:0] out;

always @(negedge reset or posedge clk) begin
    if (~reset)
        out <= 0;
    else begin
        if (out == 4'b1001)
            out <= 4'b0000;
        else
            out <= out + 1;
    end
end
endmodule
```

接 BCD 译码器后顶层单元代码如下：

```
module top (reset, clk, leds, EN);
input reset;
input clk;
output [7:0] leds;
output EN;

wire [7:0] leds;
wire [3:0] bcd;
wire EN;

counter cnt (reset, clk, bcd);
BCD7 bcd7 (bcd, leds, EN);

endmodule
```

testbench 代码如下，主要测试了异步的复位功能与两个周期的计数功能：

```
`timescale 1ns/1ps
`define PERIOD 10
```

```

module top_tb;
reg reset;
reg clk;
wire [7:0] leds;
wire EN;

initial begin
    reset <= 0;
    clk <= 0;
end

initial fork
    #150 reset <= 1;
    #380 reset <= 0;
    forever
        #(`PERIOD/2) clk <= ~clk;
    #400 $finish;
join

top counter_top (
        .reset (reset),
        .clk (clk),
        .leds (leds),
        .EN (EN)
    );

endmodule
```

管脚约束文件从略，详见附件代码，下同。

3.3 有限状态机序列检测器

有限状态机代码如下，使用三段式有限状态机设计，为 Mealy 机。

```
module fsm (clk, reset, in, out);
input clk, reset, in;
output out;
reg out;

```

```
reg [2:0] current_state, next_state;
localparam A=3'b000;
localparam B=3'b001;
localparam C=3'b011;
localparam D=3'b110;
localparam E=3'b101;
localparam F=3'b010;

always @(negedge reset or posedge clk) begin
    if (~reset)
        current_state <= A;
    else
        current_state <= next_state;
end

always @(current_state or in) begin
    case (current_state)
        A: next_state <= in ? B : A;
                B: next_state <= in ? B : C;
                C: next_state <= in ? D : A;
                D: next_state <= in ? B : E;
                E: next_state <= in ? F : A;
                F: next_state <= in ? B : E;
                default: next_state <= A;
    endcase
end

always @(negedge reset or posedge clk) begin
        if (~reset)
        out <= 0;
    else
                out <= (current_state == F && in) ? 1 : 0;
end
endmodule
```

testbench 代码如下，为连续两段示例输入序列。

```
`timescale 1ns/1ps
`define PERIOD 10

```

```
4  module fsm_tb;
5  reg reset;
6  reg clk;
7  reg in;
8  wire out;
9
10 initial begin
11     reset <= 0;
12     clk <= 0;
13 end
14
15 initial fork
16     #(`PERIOD) reset <= 1;
17     forever
18         #(`PERIOD/2) clk <= ~clk;
19 join
20
21 initial begin
22     #(`PERIOD) in = 0;
23     #(`PERIOD) in = 0;
24     #(`PERIOD) in = 1;
25     #(`PERIOD) in = 0;
26     #(`PERIOD) in = 1;
27     #(`PERIOD) in = 0;
28     #(`PERIOD) in = 1;
29     #(`PERIOD) in = 1;
30     #(`PERIOD) in = 0;
31     #(`PERIOD) in = 1;
32     #(`PERIOD) in = 0;
33     #(`PERIOD) in = 1;
34     #(`PERIOD) in = 1;
35     #(`PERIOD) in = 1;
36     #(`PERIOD) in = 0;
37     #(`PERIOD) in = 0;
38     #(`PERIOD) in = 0;
39     #(`PERIOD) in = 1;
40     #(`PERIOD) in = 0;
41     #(`PERIOD) in = 1;
42     #(`PERIOD) in = 0;
```

```
43      #(`PERIOD) in = 1;
44      #(`PERIOD) in = 1;
45      #(`PERIOD) in = 0;
46      #(`PERIOD) in = 0;
47      #(`PERIOD) in = 0;
48      #(`PERIOD) in = 0;
49      #(`PERIOD) in = 1;
50      #(`PERIOD) in = 0;
51      #(`PERIOD) in = 1;
52      #(`PERIOD) in = 0;
53      #(`PERIOD) in = 1;
54      #(`PERIOD) in = 1;
55      #(`PERIOD) in = 0;
56      #(`PERIOD) in = 1;
57      #(`PERIOD) in = 0;
58      #(`PERIOD) in = 1;
59      #(`PERIOD) in = 1;
60      #(`PERIOD) in = 1;
61      #(`PERIOD) in = 0;
62      #(`PERIOD) in = 0;
63      #(`PERIOD) in = 0;
64      #(`PERIOD) in = 1;
65      #(`PERIOD) in = 0;
66      #(`PERIOD) in = 1;
67      #(`PERIOD) in = 0;
68      #(`PERIOD) in = 1;
69      #(`PERIOD) in = 1;
70      #(`PERIOD) in = 0;
71      #(`PERIOD) in = 0;
72      $finish;
73  end
74
75  fsm t_fsm (
76          .clk(clk),
77          .reset(reset),
78          .in(in),
79          .out(out)
80      );
81
```

```
82 endmodule
```

3.4 移位寄存器序列检测器

移位寄存器代码如下，这是一个右移串行输入、并行输出寄存器，复位时存储数据清零：

```
1  module shift_register (clk, reset, in, out);
2  input clk, reset, in;
3  output [5:0] out;
4  reg [5:0] out;
5
6  always @(negedge reset or posedge clk) begin
7      if (~reset)
8          out <= 0;
9      else
10         out <= {in, out[5:1]};
11 end
12 endmodule
```

加入比较寄存器值的组合逻辑后，顶层模块代码如下。由于是右移寄存器，因此存储的先前序列是反向存储的，与待匹配序列反向匹配。

```
1  module detector (clk, reset, in, out, register);
2  input clk, reset, in;
3  output out;
4  output [5:0] register;
5
6  reg out;
7  wire [5:0] register;
8
9  shift_register sr(clk, reset, in, register);
10
11 always @(register) begin
12         if (register[5:0] == 6'b110101) // register stores reverse of 101011
13                 out <= 1;
14         else
15                 out <= 0;
16 end
17 endmodule
```

testbench 与有限状态机序列检测器类似，同样为连续两段示例输入序列，除例化序列检测器模块不同外其它完全一致，此处略去具体代码。

4　仿真结果与分析

4.1　加法计数器

加法计数器仿真结果如图 5所示，能够正确地执行异步复位、加法计数及 BCD 码译码功能。

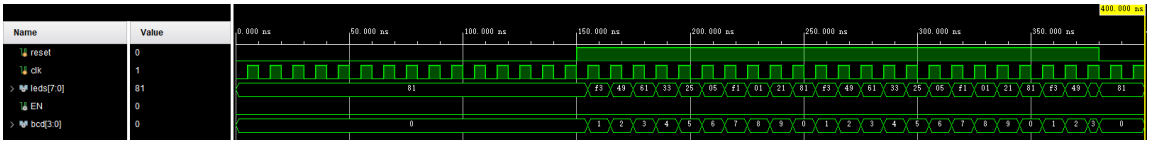

图 5: 加法计数器仿真结果

电路功能单元占用 FPGA 逻辑资源情况如图 6所示，共占用了 7 个 LUT 和 4 个寄存器。其中加法计数器占用了 3 个 LUT 和 4 个寄存器，其余 4 个 LUT 由 BCD 译码单元占用。

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
top	7	4	3	7	11	1
cnt (counter)	3	4	2	3	0	0
bcd7 (BCD7)	4	0	1	4	0	0

图 6: 加法计数器资源报告

加法计数器电路图如图 7所示，含 5 个 LUT（实现中多个 LUT 可用一个 6 输入 LUT 实现，因而实际 LUT 数目更少）和 3 个触发器。

其中四个带异步复位的 D 触发器（FDCE）接在 LUT 后用以与时钟沿同步，在每个时钟上升沿计数。两个 1 输入 LUT 配置均为 2'b01，即输入 0 时输出 1，输入 1 时输出 0，相当于非门。复位信号经过该非门接入各触发器复位端，低电平时实现 FDCE 高电平复位功能。out_reg[0] 的 D 触发输入为 out[0] 取反，即 out[0] 每个时钟周期取反，符合计数器。out[1]_i_1 配置为 16'h2C3C=16'b0010_1100_0011_1100，输入有对应 I0=out[2]，I1=out[1]，I2=out[0]，I3=out[3]。out[2]_i_1 配置为 8'h6A=8'b0110_1010，输入有对应 I0=out[2]，I1=out[1]，I2=out[0]。out[3]_i_1 配置为 16'h6F80=16'b0110_1111_1000_0000，输入有对应 I0=out[2]，I1=out[1]，

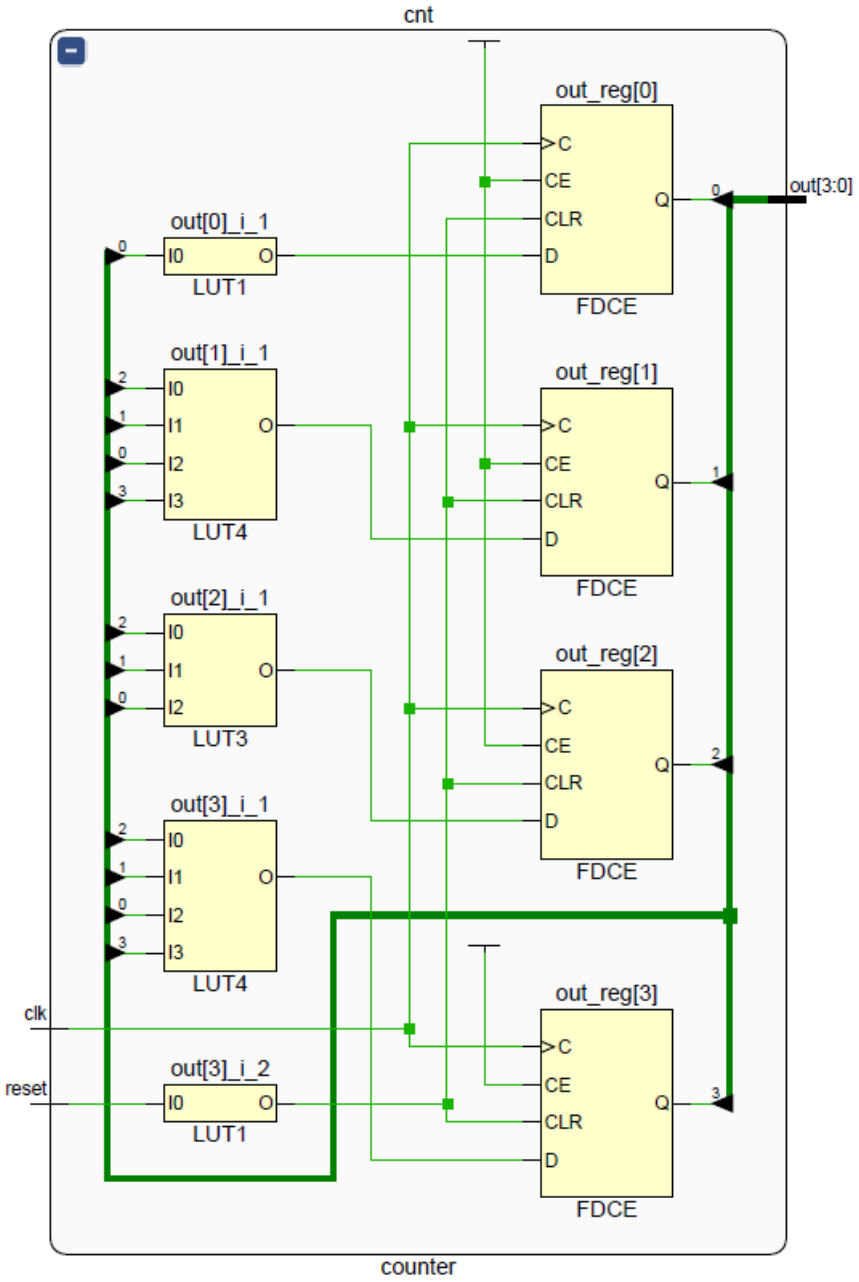
cnt
out_reg[0]
C
CE
CLR
D
Q
FDCE
out[3:0]
out[0]_i_1
I0
O
LUT1
out[1]_i_1
I1
I2
I3
LUT4
out_reg[1]
out[2]_i_1
LUT3
out_reg[2]
out[3]_i_1
out_reg[3]
clk
reset
out[3]_i_2
counter

图 7: 加法计数器电路图

I2=out[0]，I3=out[3]。表 3说明 LUT 配置实现计数功能，其中 out 次态表示下一个时钟上升沿到来后触发输出的值，即当前 LUT 的输出，可见确实实现加法计数器功能，9 的次态为 0，超过 9 的状态也会递增直至溢出得 0 自启动。

out	out[1]_i_1 输入	out[2]_i_1 输入	out[3]_i_1 输入	out 次态
4'b0000	4'b0000	3'b000	4'b0000	4'b0001
4'b0001	4'b0100	3'b100	4'b0100	4'b0010
4'b0010	4'b0010	3'b010	4'b0010	4'b0011
4'b0011	4'b0110	3'b110	4'b0110	4'b0100
4'b0100	4'b0001	3'b001	4'b0001	4'b0101
4'b0101	4'b0101	3'b101	4'b0101	4'b0110
4'b0110	4'b0011	3'b011	4'b0011	4'b0111
4'b0111	4'b0111	3'b111	4'b0111	4'b1000
4'b1000	4'b1000	3'b000	4'b1000	4'b1001
4'b1001	4'b1100	3'b100	4'b1100	4'b0000
4'b1010	4'b1010	3'b010	4'b1010	4'b1011
4'b1011	4'b1110	3'b110	4'b1110	4'b1100
4'b1100	4'b1001	3'b001	4'b1001	4'b1101
4'b1101	4'b1101	3'b101	4'b1101	4'b1110
4'b1110	4'b1011	3'b011	4'b1011	4'b1111
4'b1111	4'b1111	3'b111	4'b1111	4'b0000

表 3: 加法计数器 LUT 配置表

4.2 有限状态机序列检测器

有限状态机序列检测器仿真结果如图 8所示，正确检出了所有 101011 序列，包括重叠序列。此仿真中输入在时钟下降沿改变，实际输入在时钟上升沿同步改变也正确。

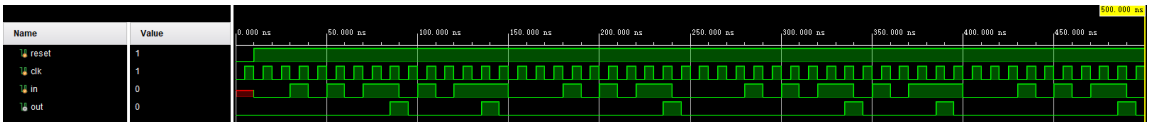

图 8: 有限状态机序列检测器仿真结果

电路功能单元占用 FPGA 逻辑资源情况如图 9所示，共占用了 3 个 LUT 和 4 个寄存器。

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
N fsm	3	4	2	3	4	1

图 9: 有限状态机序列检测器资源报告

电路图如图 10所示，含 4 个 LUT 和 3 个触发器。有限状态机状态如图 11所示，被修改为顺序编码而非代码中所给人工编码，因为使用 LUT 查找次态时，不同编码不影响使用 LUT 数量（同时这个状态机中组合逻辑较少，使用独热码需要 6 个触发器及额外的 LUT，较浪费资源）。其中 3 个触发器（FDCE）用以存储状态的 3 位，复位信号同加法计数器一样，经过 1 输入 LUT 求反送入各触发器，还有 1 个触发器用以在 Mealy 机中隔离输入，使输出保持同步。

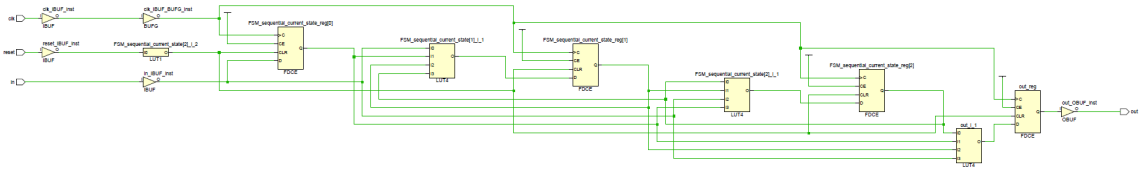

图 10: 有限状态机序列检测器电路图

```
INFO: [Synth 8-802] inferred FSM for state register 'current_state_reg' in module 'fsm'
```

State	New Encoding	Previous Encoding
A	000	000
B	001	001
C	010	011
D	011	110
E	100	101
F	101	010

```
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'sequential' in module 'fsm'
```

图 11: 有限状态机状态

FSM_sequential_current_state[1]_i_1 配置为 16'h0024=16'b0000_0000_0010_0100，输入有对应 I0=in，I1=current_state[0]（简记 FSM_sequential_current_state_reg[0] 输出，下

同)，I2=current_state[1]，I3=current_state[2]。FSM_sequential_current_state[2]_i_1 配置为 16'h0EA0=16'b0000_1110_1010_0000,输入有对应 I0=current_state[2],I1=current_state[1],I2=in，I3=current_state[0]。out_i_1 配置为 16'h0800=16'b0000_1000_0000_0000，输入有对应 I0=current_state[2]，I1=current_state[0]，I2=current_state[1]，I3=in。

FSM_sequential_current_state_reg[0] 输入 in，FSM_sequential_current_state_reg[1] 输入 FSM_sequential_current_state[1]_i_1 输出,FSM_sequential_current_state_reg[2] 输入 FSM_sequential_current_state[2]_i_1 输出。有限状态机的状态转移表如表 4所示，以 I1、I2、I3 分别表示 FSM_sequential_current_state[1]_i_1、FSM_sequential_current_state[2]_i_1 和 out_i_1 的输入，每一列两个值分别表示输入 0/1 的情形。

现态	I1	I2	I3	次态	out
3'b000(A)	4'b0000/4'b0001	4'b0000/4'b0100	4'b0000/4'b1000	3'b000(A)/3'b001(B)	0/0
3'b001(B)	4'b0010/4'b0011	4'b1000/4'b1100	4'b0010/4'b1010	3'b010(C)/3'b001(B)	0/0
3'b010(C)	4'b0100/4'b0101	4'b0010/4'b0110	4'b0100/4'b1100	3'b000(A)/3'b011(D)	0/0
3'b011(D)	4'b0110/4'b0111	4'b1010/4'b1110	4'b0110/4'b1110	3'b100(E)/3'b001(B)	0/0
3'b100(E)	4'b1000/4'b1001	4'b0001/4'b0101	4'b0001/4'b1001	3'b000(A)/3'b101(F)	0/0
3'b101(F)	4'b1010/4'b1011	4'b1001/4'b1101	4'b0011/4'b1011	3'b100(E)/3'b001(B)	0/1
3'b110	4'b1100/4'b1101	4'b0011/4'b0111	4'b0101/4'b1101	3'b000(A)/3'b101(F)	0/0
3'b111	4'b1110/4'b1111	4'b1011/4'b1111	4'b0111/4'b1111	3'b100(E)/3'b001(B)	0/0

表 4: 有限状态机 LUT 配置表

可见更改为顺序编码的状态均符合先前所确定的状态转移与输出规则，仅是编码做了改变，而不存在的状态则未必如程序中所设定默认跳转回初态，但仍然输出 0，且在一个时钟周期后均跳转回已存在状态，可自启动。

4.3 移位寄存器序列检测器

移位寄存器序列检测器仿真结果如图 12所示，正确检出序列，移位寄存器反向存储的是先前六个输入。

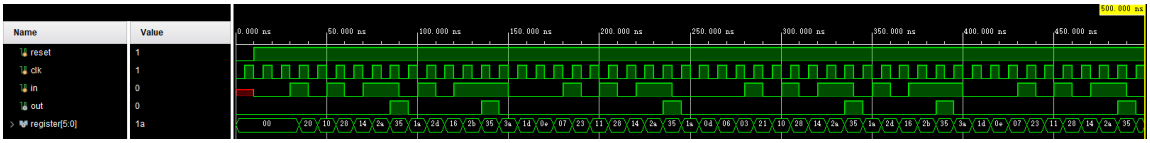

图 12: 移位寄存器序列检测器仿真结果

电路功能单元占用 FPGA 逻辑资源情况如图 13所示，共占用了 2 个 LUT 和 11 个寄存

器。

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
detector	2	11	2	2	10	1
sr (shift_register)	2	11	2	2	0	0

图 13: 移位寄存器序列检测器资源报告

电路图如图 14所示，为 6 个 D 触发器依次相连，从而在每个时钟上升沿移位，空位由输入补齐，最后的 6 输入 LUT 用来与待测序列比较，是独热的 LUT（配置 64'h0000_0000_4000_0000）。

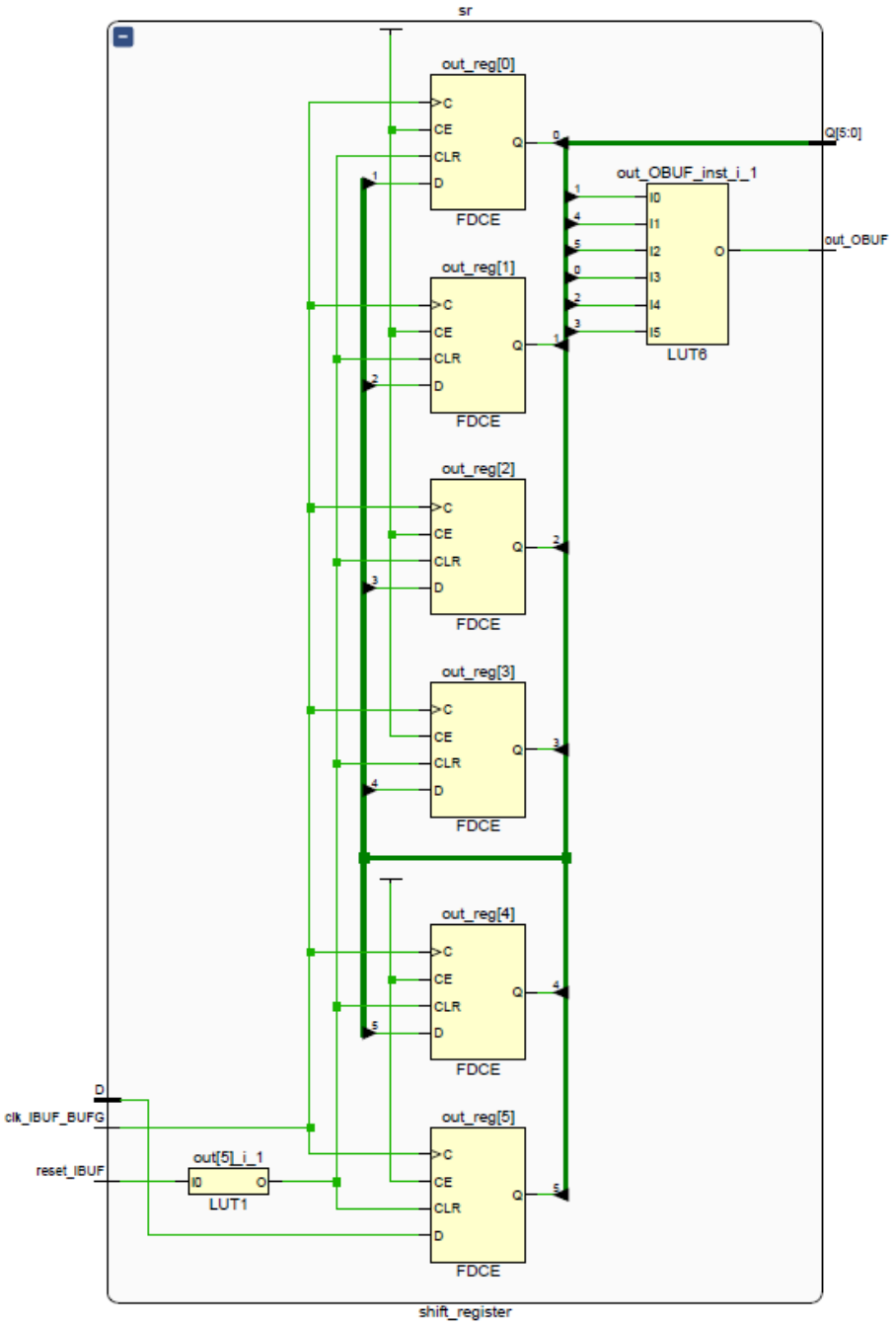

图 14: 移位寄存器序列检测器电路图