实验报告

BobAnkh

实验一 4bits十进制同步加法计数器

1、实验目的：

设计一个具有异步复位控制的 4bits 十进制同步加法计数器，此计数器输出的 BCD 码通过译码电路送入七段数码管显示

2、设计方案：

将整个设计拆分成两个子模块进行设计，第一个模块为设计的具有异步复位功能的十进制同步加法计数器，每一个时钟周期输出对应的 4-bit 的 bcd 码；第二个模块直接采用之前的七段译码器的设计模块，并加入了使能控制。两个模块分别设计完成后，直接串接在一起实现本实验所需要的功能。

3、关键代码：

这里仅给出设计代码，xdc 约束文件和 testbench 文件可以见对应附件。

计数器模块：

```
module cnt(
    input clk,
    input reset,
    output reg [3:0] cnt_out
);
always @(negedge reset or posedge clk)
begin
    if(~reset)
        cnt_out <= 4'b0000;
    else begin
        if (cnt_out==4'b1001) cnt_out<=4'b0000;
        else cnt_out<=cnt_out+1;
    end
end
endmodule
```

七段译码器模块：

```
module bcd7(
    input [3:0] din,
    input en,
```

```
    output reg [6:0] dout
);

always @(*)
begin
    if (en==1'b0) dout<=7'b0;
    else if(din==4'h0) dout<=7'b0111111;
    else if(din==4'h1) dout<=7'b0000110;
    else if(din==4'h2) dout<=7'b1011011;
    else if(din==4'h3) dout<=7'b1001111;
    else if(din==4'h4) dout<=7'b1100110;
    else if(din==4'h5) dout<=7'b1101101;
    else if(din==4'h6) dout<=7'b1111101;
    else if(din==4'h7) dout<=7'b0000111;
    else if(din==4'h8) dout<=7'b1111111;
    else if(din==4'h9) dout<=7'b1101111;
    else dout<=7'b0;
end
endmodule
```

顶层文件：

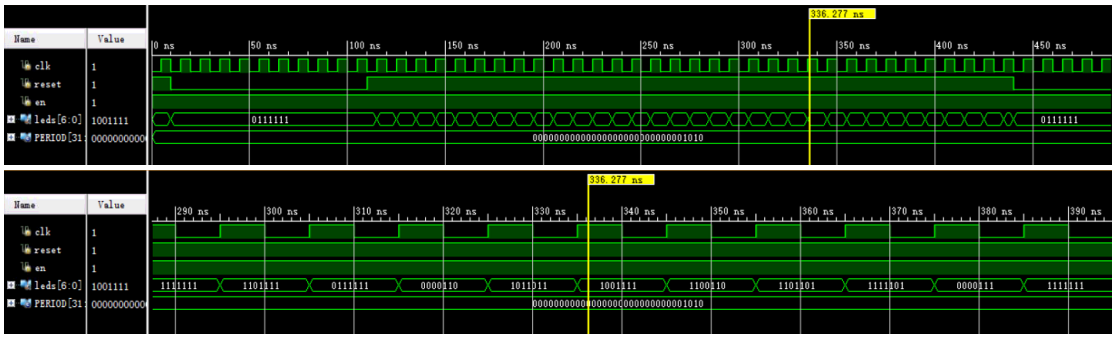
```
`timescale 1ns / 1ps
module top (
    input clk,
    input reset,
    input en,
    output wire [6:0] leds
);

wire [3:0] bcd;
cnt bcdcounter (.clk(clk), .reset(reset), .cnt_out(bcd));
bcd7 bcd27seg (.din(bcd),.en(en), .dout(leds));

endmodule
```

4、文件清单：

在文件夹“加法计数器代码”中，在子文件夹“设计代码”中放置了 cnt.v，bcd7.v 和 top.v，在子文件夹“testbench 代码和约束文件”中放置了 cnt_tb.v 和 cns.xdc

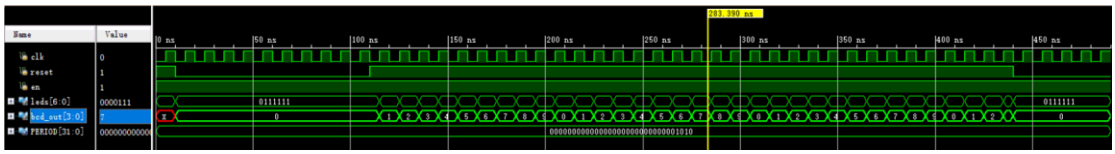
5、仿真结果及分析：

前仿真：



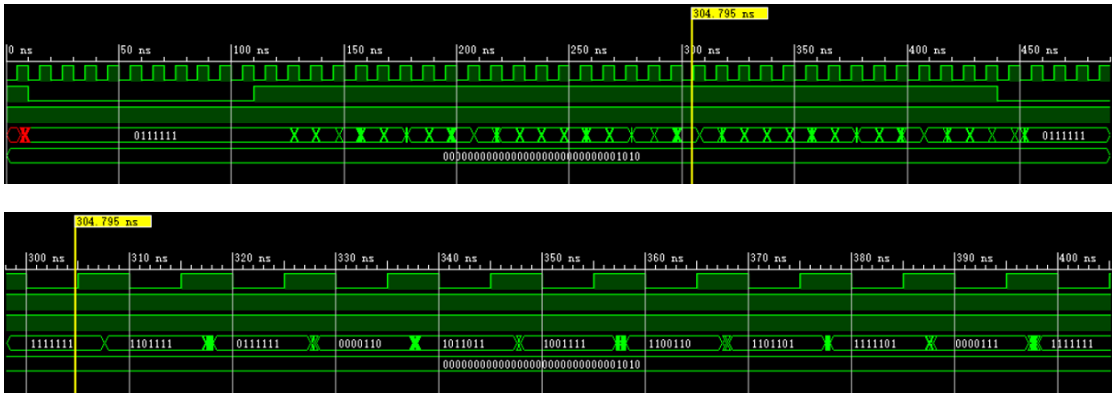
从整体上可以看到，当 reset 信号置为 0 时，整体输出为 0111111，也就是 0，可以看出实现了异步复位的功能；放大到局部之后，以此图为例来进行说明，可以看到整体最终的输出以每个时钟周期进行一次变化而形成了一个 1111111->1101111->0111111->0000110->1011011->1001111->1100110->1101101->1111101->0000111->1111111 这样一个循环，通过对应我们可以知道这里也就是从 8->9->0->1->2->3->4->5->6->7->8 这样的一个循环，可以看出实现了设计的功能，能够从 0 顺序计数到 9 并回到 0。

为了更直观的验证，再次添加了输出 4 位 bcd 计数进行仿真：

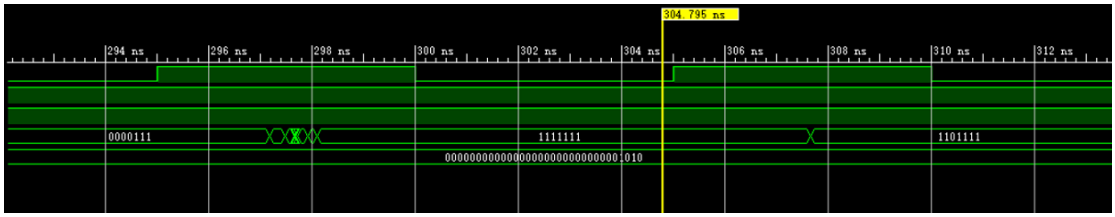


从整体上可以看到，当 reset 信号置为 0 时，bcd_cnt 输出为 0，同时，可以看到其输出以每个时钟周期进行一次变化而形成了一个 0->1->2->3->4->5->6->7->8->9->0 这样的一个反复循环，可以直观地看出实现了设计的功能。

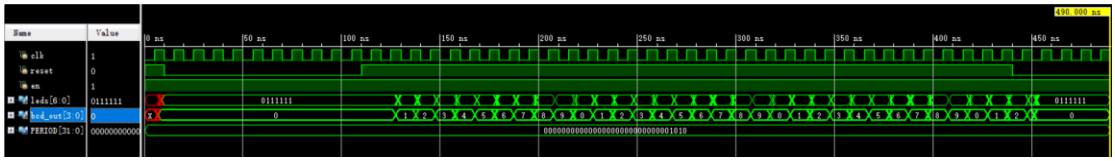
后仿真（时序仿真）：



虽然存在一些时延和变化时的毛刺，但是通过输出的数的变化，通过和前仿真部分同样的分析过程可以看出它实现了设计的功能，计数逻辑功能正确，也可以做到异步复位。
细看会发现有一些毛刺和时延：



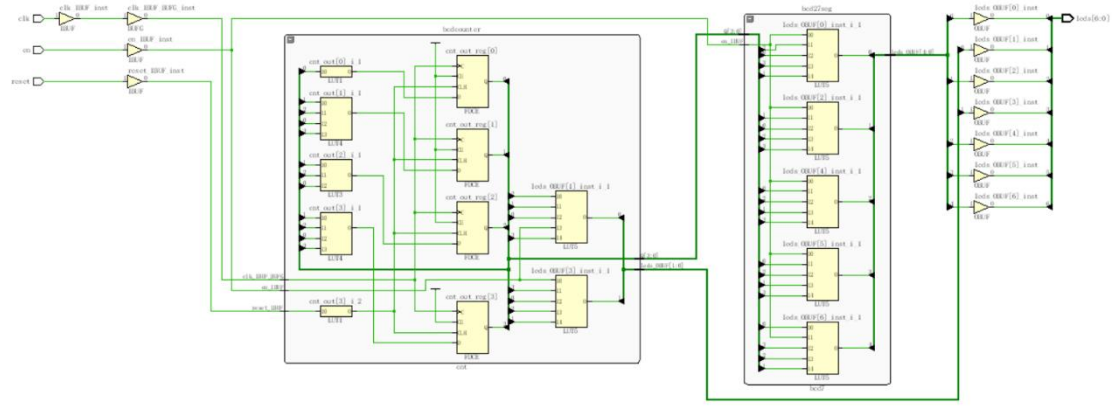
为了更直观的验证，再次添加了输出 4 位 bcd 计数进行仿真：



可以直观地看到，虽然有一定的延时和毛刺，但是计数的逻辑功能正确且可以做到异步复位。

6、综合情况：

查看综合之后的电路原理图如下：



FPGA 逻辑资源情况：

直接数电路图上的逻辑资源使用情况为：

bcd 计数器使用了 7 个 LUT 和 4 个触发器，而七段译码器使用了 5 个 LUT，一共为 12 个 LUT 和 4 个触发器。在后续验证功能的过程中还发现，原理图中 bcd 计数器中有两个 LUT 实际上是实现的七段译码器的功能，只是将其放置在计数器模块中，原因还有待后续学习和深入研究。

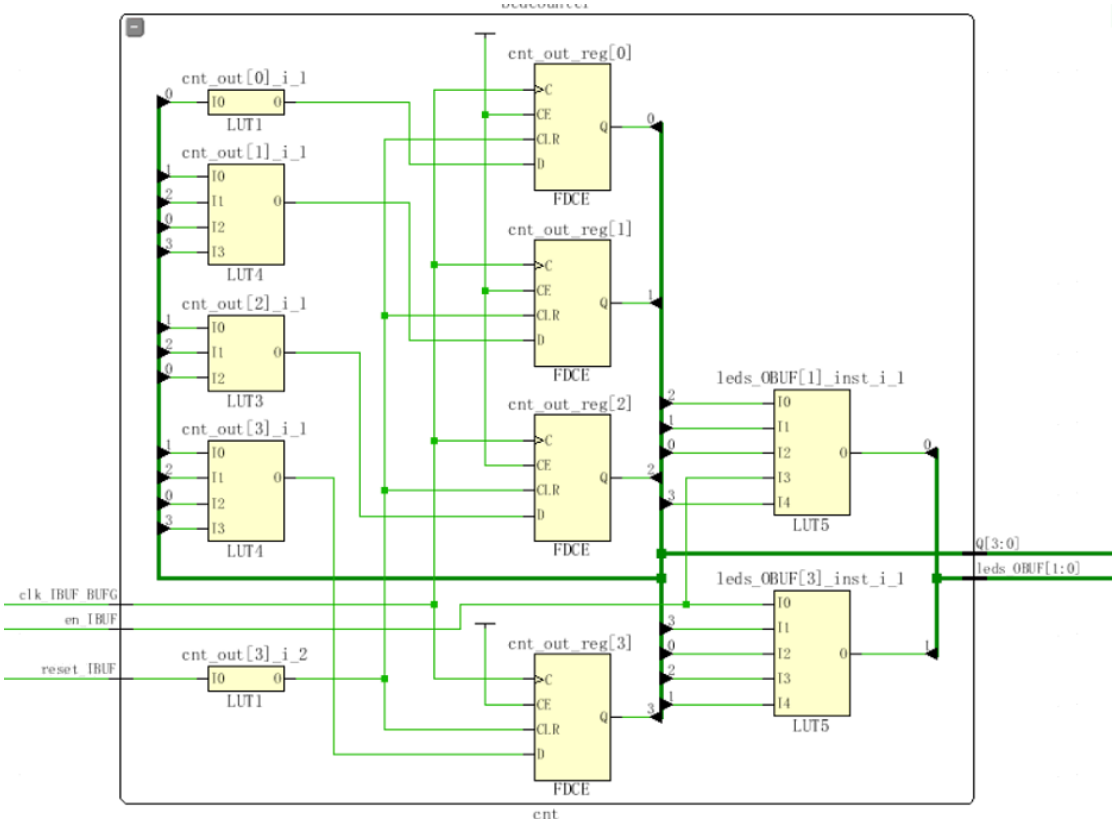
直接查看使用报告的结果为：

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (210)	BUFGCTRL (32)
top	7	4	2	7	7	14	1
bcd27seg (bcd7)	3	0	1	3	3	0	0
bcdcounter (cnt)	4	4	2	4	4	0	0

可以看到，LUT 比电路原理图上数出来的要少，推测应当是在实际实现的时候做了一定的优化。

通过查看电路图和 LUT 配置检验加法计数器的正确性：

以下是加法计数器模块的电路原理图：



这一部分仅考虑加法计数器的模块功能的实现，可以发现，右侧两个 LUT 实现的是七段译码器的部分功能，本部分分析就不加以考虑了，中间四个触发器，CLR 端都经过一个 LUT 连接 reset 信号，中间经过的 LUT 的配置字为 2'h1，很容易看出就是实现了一个异步复位的功能。

主要分析图示为 cnt_out[0-3]_i_1 的四个 LUT：

（用 BCD_OUT[0-3]代指触发器输出）

cnt_out[0]:

配置字 INIT=2'h1=01

输入为：

I0
BCD_OUT[0]

cnt_out[1]:

配置字 INIT=16'h4A5A=0100_1010_0101_1010

输入为：

I0	I1	I2	I3
BCD_OUT[1]	BCD_OUT[2]	BCD_OUT[0]	BCD_OUT[3]

cnt_out[2]:

配置字 INIT=8'h6C=0110_1100
输入为：

I0	I1	I2
BCD_OUT[1]	BCD_OUT[2]	BCD_OUT[0]

cnt_out[3]:
配置字 INIT=16'h6F80=0110_1111_1000_0000
输入为：

I0	I1	I2	I3
BCD_OUT[1]	BCD_OUT[2]	BCD_OUT[0]	BCD_OUT[3]

那么由此可以画出设计实现的加法计数器的状态转移表：

现态[3:0]	次态[3:0]
0000	**0001**
0001	**0010**
0010	**0011**
0011	**0100**
0100	**0101**
0101	**0110**
0110	**0111**
0111	**1000**
1000	**1001**
1001	**0000**
1010	1011
1011	1110
1100	**0101**
1101	1100
1110	1101
1111	**0000**

可以看到逻辑功能是正确的，0-9 的计数是正确的，计数到 9 后下一个就是 0，同时，余下的态（10-15 的数）也都能够通过几步转化变回到 10 个需要的数，能够自启动，所以加法计数器的设计是正确的。

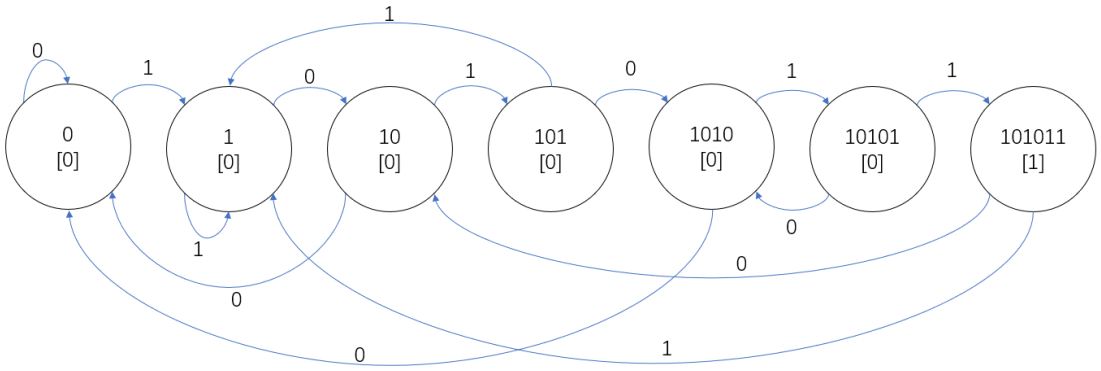
实验二 序列检测器——状态机

1、实验目的：

在连续输入的串行数据流中检测特定序列“101011”，一旦检测到一个“01011“”就输出一个宽度为 1 个时钟周期的高电平脉冲。

2、设计方案：

采用 Moore 机，按如下状态转移图进行设计（[ ]内为输出，箭头线上为输入）：



3、关键代码：

这里仅给出设计代码，xdc 约束文件和 testbench 文件可以见对应附件。

状态机：

```
`timescale 1ns / 1ps

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

reg [2:0] current_state, next_state;

always @(negedge reset or posedge clk)
begin
    if(~reset)  current_state <= 3'b0;
    else current_state <= next_state;
end

always @(*) begin
case(current_state)
```

```
    3'd0: begin
        seq_out <= 0;
        if(in) next_state <= 3'd1;
        else next_state <= 3'd0;
    end
    3'd1: begin
        seq_out <= 0;
        if(in) next_state <= 3'd1;
        else next_state <= 3'd2;
    end
    3'd2: begin
        seq_out <= 0;
        if(in) next_state <= 3'd3;
        else next_state <= 3'd0;
    end
    3'd3: begin
        seq_out <= 0;
        if(in) next_state <= 3'd1;
        else next_state <= 3'd4;
    end
    3'd4: begin
        seq_out <= 0;
        if(in) next_state <= 3'd5;
        else next_state <= 3'd0;
    end
    3'd5: begin
        seq_out <= 0;
        if(in) next_state <= 3'd6;
        else next_state <= 3'd4;
    end
    3'd6: begin
        seq_out <= 1;
        if(in) next_state <= 3'd1;
        else next_state <= 3'd2;
    end
    default: begin
        seq_out <= 0;
        next_state <= 3'b0;
    end
endcase
end

endmodule
```

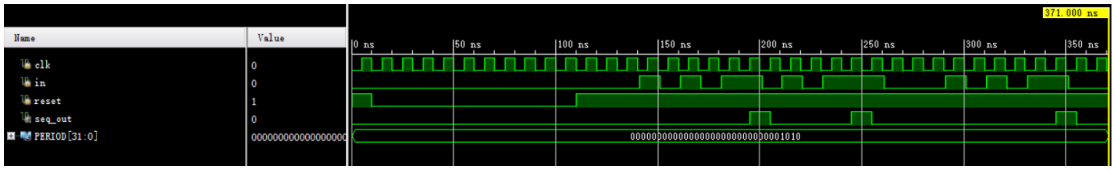
4、文件清单：

在文件夹“序列检测器_状态机代码”中，在子文件夹“设计代码”中放置了 fsm.v，在子文件夹“testbench 代码和约束文件”中放置了 fsm_tb.v 和 seq_fsm.xdc

5、仿真结果及分析：

本次仿真的输入采用了题干中所提供的 24 位输入：001010110101110001010110 0

前仿真：

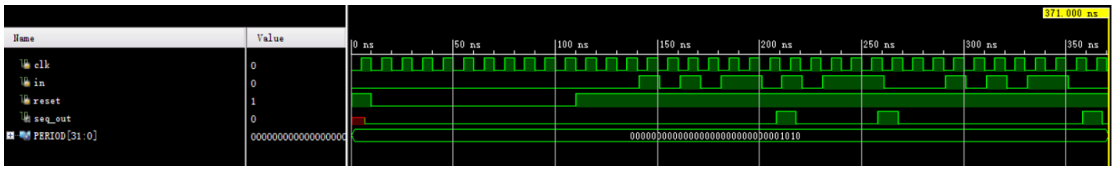


Tcl console:

```
# run 1000ns
Simulation Start!
Test begin:  00000000100001000000000010  test finish!
$finish called at time : 371 ns : File "C:/Users/lenovo/project_3/project_3.srcs/sim_1/new/fsm_tb.v" Line 64
```

可以看到输出完全符合设计要求，输出结果为：00000000100001000000000010，与题干所给结果一致。

后仿真（时序仿真）：

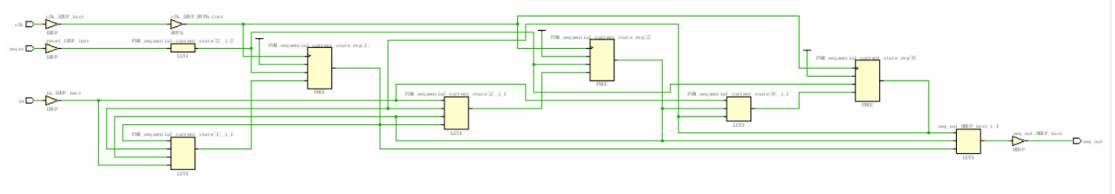


Tcl console:

```
# run 1000ns
Simulation Start!
Test begin:  00000000100001000000000001  test finish!
$finish called at time : 371 ns : File "C:/Users/lenovo/project_3/project_3.srcs/sim_1/new/fsm_tb.v" Line 64
```

可以看到因为延时，所以输出信号的形状与前仿一样，但是有一个向后错位，tcl 控制台的输出也因此前面多了一个 0，但是总体的序列检测的功能是正确的。

6、综合情况：



FPGA 逻辑资源情况：

直接数电路图上的逻辑资源使用情况为：

一共使用了 5 个 LUT 和 3 个触发器。

直接查看使用报告的结果为：

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (210)	BUFGCTRL (32)
fsm	5	3	2	5	5	4	1

可以看到，LUT 和触发器数量和电路原理图是一致的。

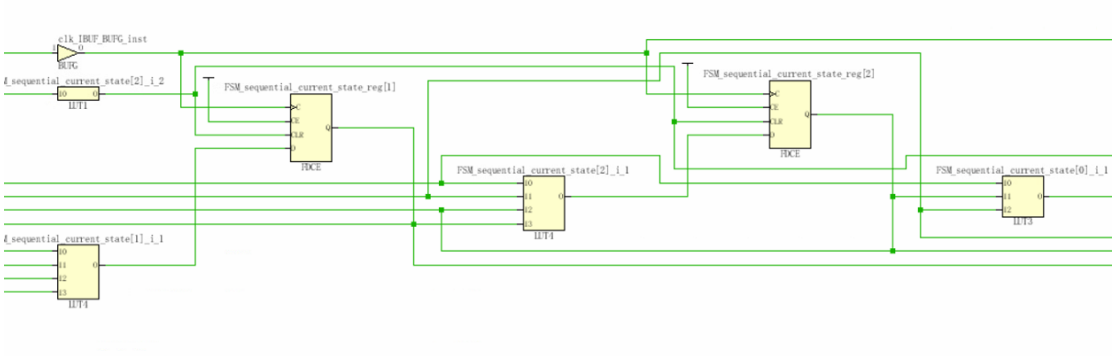
分析状态机所使用的编码方式：

查看综合报告：

```
INFO: [Synth 8-802] inferred FSM for state register 'current_state_reg' in module 'fsm'
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'sequential' in module 'fsm'
```

发现综合报告中直接说明了状态机采用的是顺序编码。

再直接结合电路原理图进行验证：



考察 current_state[2:0]的状态转移的 LUT 的配置字：

current_state [0]:

配置字 INIT=8'h2A=0010_1010

输入为：

I0	I1	I2
in	state[2]	state[0]

current_state [1]:

配置字 INIT=16'h4A5A=0100_1010_0101_1010

输入为：

I0	I1	I2	I3
state[1]	state[0]	state[2]	in

current_state [2]:

配置字 INIT=8'h6C=0110_1100

输入为：

I0	I1	I2	I3
in	state[0]	state[2]	state[1]

由此可以画出状态转移表：

现态[2:0]	次态[2:0]	
	输入 in=0	输入 in=1
000	000	001
001	010	001
010	000	011
011	100	001
100	000	101
101	100	110
110	010	001
111	000	000

可以看到状态编码是把设计中 1 和 101011 两个状态合并后的顺序编码

实验二 序列检测器——移位寄存器

1、实验目的：

在连续输入的串行数据流中检测特定序列“101011”，一旦检测到一个“01011“”就输出一个宽度为 1 个时钟周期的高电平脉冲。

2、设计方案：

每次将最高位移出，将输入补入最低位，具体采用左移位寄存器的方式进行设计，每次上升沿取上一次 6 位中的低 5 位作为新的高 5 位，把输入作为最低位。完成移位后检测 6 位数，对应不同的结果输出对应的 0 或 1。

3、关键代码：

这里仅给出设计代码，xdc 约束文件和 testbench 文件可以见对应附件。

移位寄存器：

```
`timescale 1ns / 1ps

module shift(
    input clk,
    input in,
    input reset,
    output reg seq_out
    );

reg [5:0] current_state, next_state;

always @(negedge reset or posedge clk)
begin
    if(~reset) current_state <= 0;
    else current_state <= next_state;
end

always @(*) begin
    if(current_state==6'b101011) seq_out <= 1;
    else seq_out <= 0;
    next_state <= {current_state[4:0], in};
end

endmodule
```

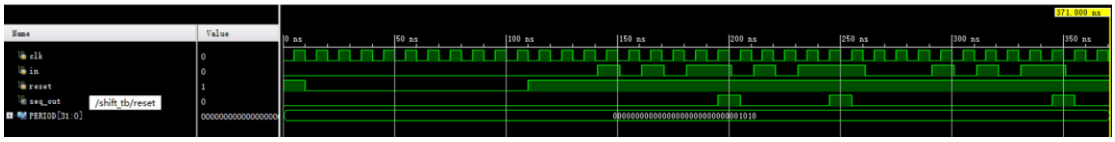
4、文件清单：

在文件夹“序列检测器_状态机代码”中，在子文件夹“设计代码”中放置了 shift.v，在子文件夹“testbench 代码和约束文件”中放置了 shift_tb.v 和 shift_cns.xdc

5、仿真结果及分析：

本次仿真的输入采用了题干中所提供的 24 位输入：001010110101110001010110O

前仿真：

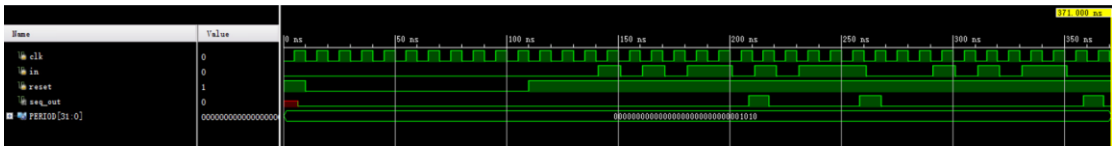


Tcl Console:

```
# run 1000ns
Simulation Start!
Test begin:  00000000100001000000000010  test finish!
$finish called at time : 371 ns : File "C:/Users/lenovo/project_4/project_4.srcs/sim_1/new/shift_tb.v" Line 64
```

可以看到输出完全符合设计要求，输出结果为：00000000100001000000000010，与题干所给结果一致。

后仿真（时序仿真）：

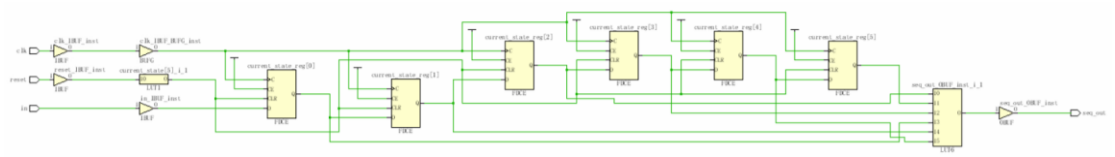


Tcl Console:

```
# run 1000ns
Simulation Start!
Test begin:  00000000010000100000000001  test finish!
$finish called at time : 371 ns : File "C:/Users/lenovo/project_4/project_4.srcs/sim_1/new/shift_tb.v" Line 64
```

可以看到因为延时，所以输出信号的形状与前仿一样，但是有一个向后错位，tcl 控制台的输出也因此前面多了一个 0，但是总体的序列检测的功能是正确的。

6、综合情况：



FPGA 逻辑资源情况：

直接数电路图上的逻辑资源使用情况为：

一共使用了 2 个 LUT 和 6 个触发器。

直接查看使用报告的结果为：

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (210)	BUFGCTRL (32)
shift	2	6	2	2	6	4	1

可以看到，LUT 和触发器数量和电路原理图是一致的。