HOMEWORK1

备注：手正面高电平输入，手背面低电平输入。

输出 00 表示 A 胜出，01 表示 B 胜出，10 表示 C 胜出，11 表示打平手。

Source code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity game is
    Port ( p_a : in STD_LOGIC;
           p_b : in STD_LOGIC;
           p_c : in STD_LOGIC;
           winner : out STD_LOGIC_VECTOR (1 downto 0));
end game;

architecture Behavioral of game is
    signal diff:STD_LOGIC;
begin
    winner<="00" when (p_b=p_c) and (p_a/=p_b) else
            "01" when (p_a=p_c) and (p_b/=p_a) else
            "10" when (p_a=p_b) and (p_c/=p_a) else
            "11";
end Behavioral;
```

Testbench code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity test_game is
--  Port ( );
end test_game;

architecture Behavioral of test_game is
    component game is
        Port ( p_a : in STD_LOGIC;
               p_b : in STD_LOGIC;
               p_c : in STD_LOGIC;
               winner : out STD_LOGIC_VECTOR (1 downto 0));
```

```
    end component;
    signal p_a,p_b,p_c:STD_LOGIC;
    signal winner:STD_LOGIC_VECTOR(1 downto 0);
begin
    uut:game
    port map(p_a,p_b,p_c,winner
    );
    process
        variable data:STD_LOGIC_VECTOR(2 downto 0);
    begin
        for i in 0 to 7 loop
            data:=std_logic_vector(to_unsigned(i,3));
            p_a<=data(2);p_b<=data(1);p_c<=data(0);
            wait for 100 ns;
        end loop;
    end process;
end Behavioral;
```

Simulation waveform

HOMEWORK2

Source code

ripple_counter.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ripple_counter is
    Port ( clk,res : in STD_LOGIC;
           q : out STD_LOGIC_VECTOR (3 downto 0));
end ripple_counter;

architecture Behavioral of ripple_counter is
    component d_flipflop is
        Port ( d : in STD_LOGIC;
               clk,res : in STD_LOGIC;
               q : out STD_LOGIC);
    end component;
    signal d_dff,q_dff:STD_LOGIC_VECTOR(3 downto 0);
begin
    dff0_3_for:for i in 0 to 3 generate
        dff0_if:if i=0 generate
            dff0:d_flipflop
                port map(d=>d_dff(i),clk=>clk,res=>res,q=>q_dff(i));  -
-special for clk
        end generate dff0_if;
        dff1_3_if:if i/=0 generate
            dff1_3:d_flipflop
            port map(d=>d_dff(i),clk=>q_dff(i-
1),res=>res,q=>q_dff(i));  --special for clk
        end generate dff1_3_if;
    end generate dff0_3_for;
    d_dff<=not q_dff;
    q<=q_dff;
end Behavioral;
```

d_flipflop.vhd

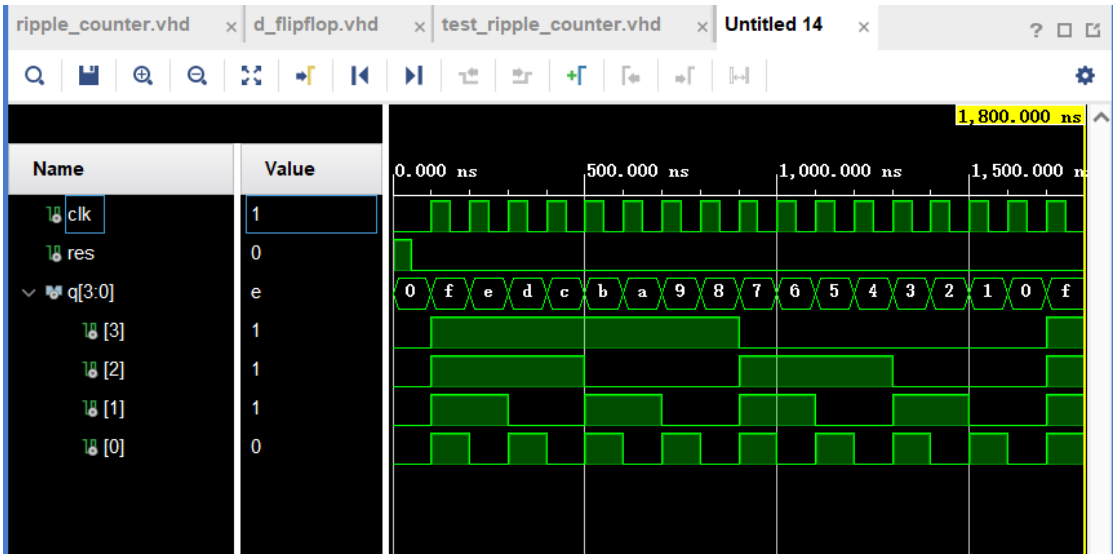
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity d_flipflop is
    Port ( d : in STD_LOGIC;
           clk : in STD_LOGIC;
           res : in STD_LOGIC;
           q : out STD_LOGIC);
```

```
end d_flipflop;

architecture Behavioral of d_flipflop is

begin
    process(clk,res)
    begin
        if res='1' then
            q<='0';
        elsif clk'event and clk='1' then
            q<=d;
        end if;
    end process;
end Behavioral;
```

Testbench code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_ripple_counter is
--  Port ( );
end test_ripple_counter;

architecture Behavioral of test_ripple_counter is
    component ripple_counter is
        Port ( clk,res : in STD_LOGIC;
               q : out STD_LOGIC_VECTOR (3 downto 0));
    end component;
    signal clk,res :STD_LOGIC;
    signal q :STD_LOGIC_VECTOR (3 downto 0);
begin
    uut:ripple_counter port map(clk,res,q);
    process
    begin
        clk<='0';
        res<='1';    wait for 50 ns;
        res<='0';    wait for 50 ns;  --initialization and reset for
counting
        while true loop    --endless clocks
        clk<='1';   wait for 50 ns;
        clk<='0';   wait for 50 ns;
        end loop;
    end process;
end Behavioral;
```

Simulation waveform



HOMEWORK3

Source code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sequence_sig_gen is
    Port ( clk : in STD_LOGIC;
           en : in STD_LOGIC;
           s : out STD_LOGIC);
end sequence_sig_gen;

architecture Behavioral of sequence_sig_gen is
    signal clk_2hz:STD_LOGIC;
    signal count_num:integer range 1 to 8;
begin
    division:process(clk,en)  --50Mhz-->2Hz,need 25M
division,12.5Mcounters.
        variable count_div:integer range 1 to 125e5;
    begin
        if en='0' then
            count_div:=1;
            clk_2hz<='0';
        elsif clk'event and clk='1'then
            if count_div=125e5 then
                count_div:=1;
                clk_2hz<=not clk_2hz;
            else
                count_div:=count_div+1;
            end if;
        end if;
    end process;

    counter:process(clk_2hz,en)
    begin
        if en='0' then
        count_num<=1;
    elsif clk_2hz'event and clk_2hz='1'then
        if count_num=8 then
            count_num<=1;
        else
            count_num<=count_num+1;
        end if;
    end if;
```

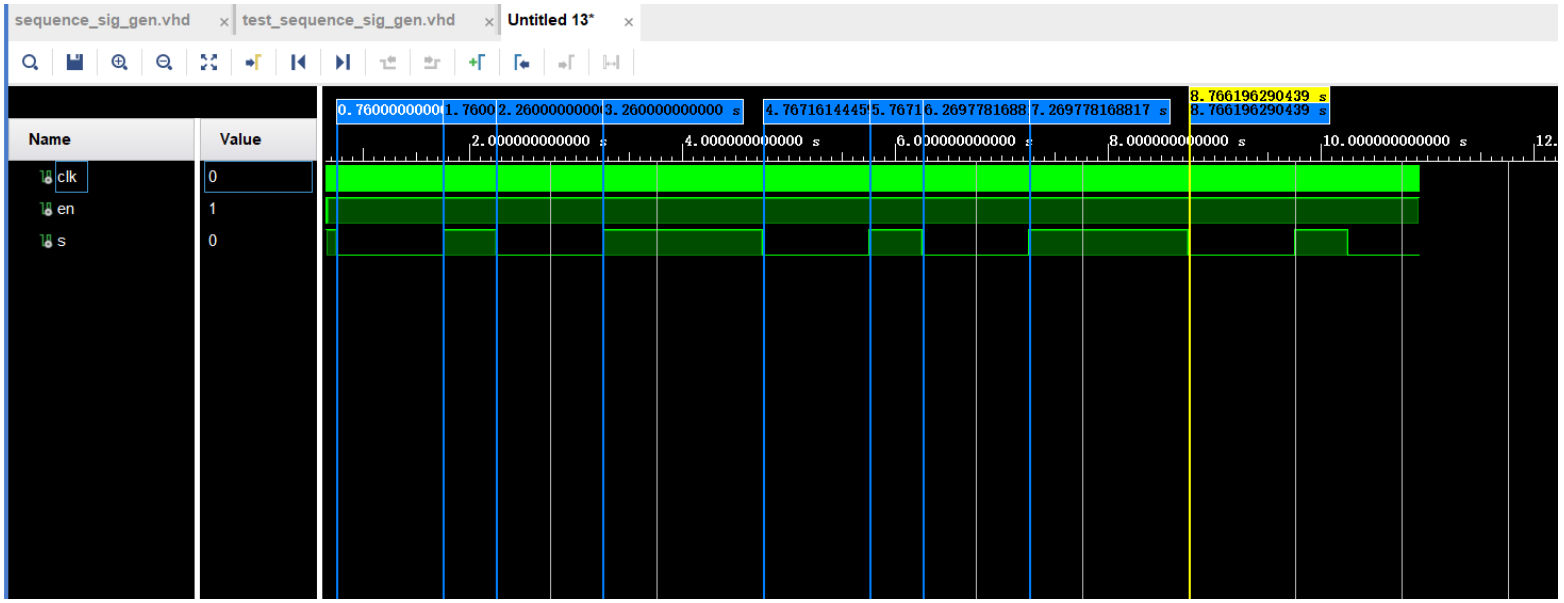
```
    end process;

    sequence:process(count_num,en)  --11001001
    begin
        if en='0' then
            s<='0';
        else
            case count_num is
                when 1|2|5|8=> s<='1';
                when 3|4|6|7=> s<='0';
                when others=>  s<='0';
            end case;
        end if;
    end process;

end Behavioral;
```

Testbench code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_sequence_sig_gen is
--  Port ( );
end test_sequence_sig_gen;

architecture Behavioral of test_sequence_sig_gen is
    component sequence_sig_gen is
        Port ( clk : in STD_LOGIC;
               en : in STD_LOGIC;
               s : out STD_LOGIC);
    end component;
    signal clk,en:STD_LOGIC;
    signal s:STD_LOGIC;
begin
    uut:sequence_sig_gen port map(clk,en,s);
    process
    begin
        clk<='0';
        en<='0';   wait for 10 ns;
        en<='1';   wait for 10 ns;
        while true loop    --endless clocks
        clk<='1';   wait for 10 ns;
        clk<='0';   wait for 10 ns;
        end loop;
```

```
    end process;
end Behavioral;
```

Simulation waveform



HOMEWORK4

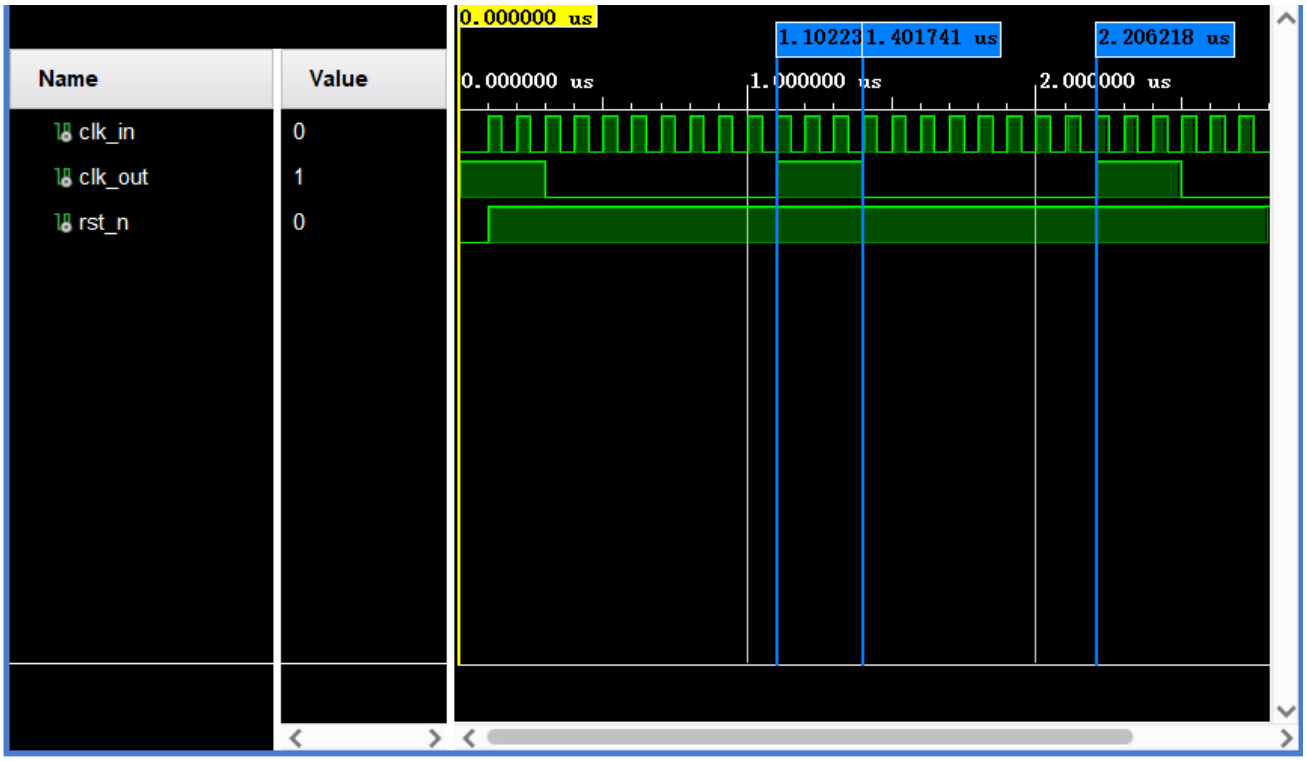
Source code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity division_11_3_11 is
    Port ( clk_in : in STD_LOGIC;
           clk_out : out STD_LOGIC;
           rst_n : in STD_LOGIC
    );
end division_11_3_11;

architecture Behavioral of division_11_3_11 is
    signal count_11:integer range 0 to 10;
begin
    process(clk_in,rst_n)
    begin
        if rst_n='0' then
            count_11<=0;
        elsif clk_in'event and clk_in='1' then
            if count_11=10 then
                count_11<=0;
            else
                count_11<=count_11+1;
            end if;
        end if;
    end process;

    process(count_11)
    begin
        if count_11<3 then
            clk_out<='1';
        else
            clk_out<='0';
        end if;
    end process;

end Behavioral;
```

Testbench code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_devision is
--  Port ( );
end test_devision;

architecture Behavioral of test_devision is
    component division_11_3_11 is
        Port ( clk_in : in STD_LOGIC;
            clk_out : out STD_LOGIC;
            rst_n : in STD_LOGIC
        );
    end component;
    signal clk_in : STD_LOGIC;
    signal clk_out : STD_LOGIC;
    signal rst_n : STD_LOGIC;
begin
    uut:division_11_3_11
        port map(
            clk_in=>clk_in,
            clk_out=>clk_out,
            rst_n=>rst_n
    );
    process
    begin
        rst_n<='0';
        clk_in<='0';
        wait for 100 ns;
        rst_n<='1';
        clk_in<='1';
        while true loop
            wait for 50 ns;
            clk_in<=not clk_in;
        end loop;
    end process;

end Behavioral;
```

Simulation waveform



HOMEWORK5

Source code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;

entity division_11_1_2 is
    Port ( clk_in : in STD_LOGIC;
           rst_n: in STD_LOGIC;
           clk_out : out STD_LOGIC);
end division_11_1_2;

architecture Behavioral of division_11_1_2 is
    signal count_rising:STD_LOGIC_VECTOR(3 downto 0);
    signal count_falling:STD_LOGIC_VECTOR(3 downto 0);
    signal count:STD_LOGIC_VECTOR(7 downto 0);
begin
    process(clk_in,rst_n)
    begin
        if rst_n='0' then
            count_rising<="0000";
        elsif clk_in'event and clk_in='1'then  --rising edge
            if count_rising="1010" then
                count_rising<="0000";
            else
                count_rising<=count_rising+1;
            end if;
        end if;
    end process;

    process(clk_in,rst_n)
    begin
        if rst_n='0' then
            count_falling<="0000";
        elsif clk_in'event and clk_in='0'then  --falling edge
            if count_falling="1010" then
                count_falling<="0000";
            else
                count_falling<=count_falling+1;
            end if;
        end if;
    end process;
```

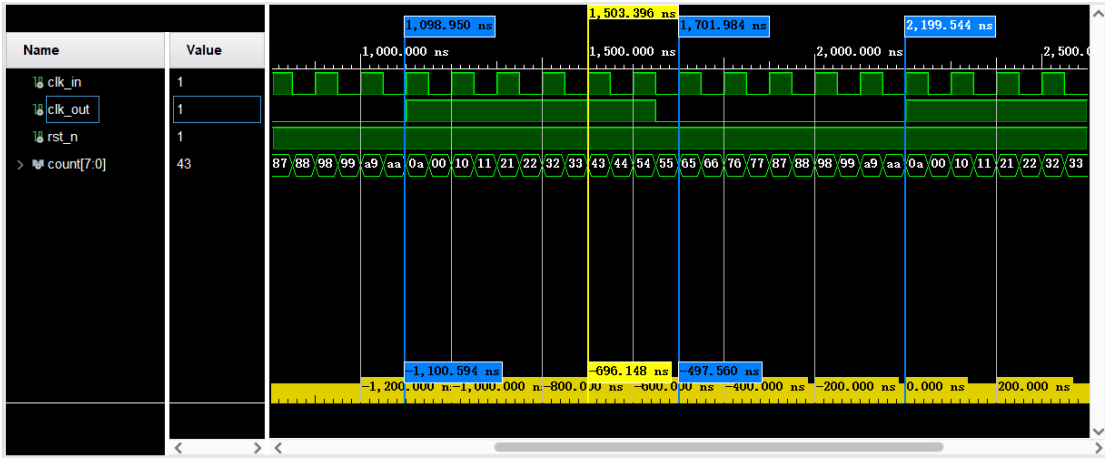
```
    count<=count_rising & count_falling;

    process(count,rst_n)
    begin
        if rst_n='0' then
            clk_out<='0';
        elsif count<"01010101" then
            clk_out<='1';
        else
            clk_out<='0';
        end if;
    end process;

--    s_div<='1' when count<"001001" else
--       '0';

end Behavioral;
```

Testbench code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_devision is
--  Port ( );
end test_devision;

architecture Behavioral of test_devision is
    component division_11_1_2 is
        Port ( clk_in : in STD_LOGIC;
            rst_n : in STD_LOGIC;
            clk_out : out STD_LOGIC
        );
    end component;
    signal clk_in : STD_LOGIC;
    signal clk_out : STD_LOGIC;
    signal rst_n : STD_LOGIC;
begin
    uut:division_11_1_2
        port map(
            clk_in=>clk_in,
            clk_out=>clk_out,
            rst_n=>rst_n
    );
    process
```

```
    begin
        rst_n<='0';
        clk_in<='0';
        wait for 100 ns;
        rst_n<='1';
        clk_in<='1';
        while true loop
            wait for 50 ns;
            clk_in<=not clk_in;
        end loop;
    end process;

end Behavioral;
```

Simulation waveform



HOMEWORK6

Moore 机

Moore--Source code

```
--Moore Machine
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sequ_scan is
    Port ( data : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst_n : in STD_LOGIC;
           check : out STD_LOGIC);
end sequ_scan;

architecture Behavioral of sequ_scan is
    --type state_enum is (s1,s2,s3,s4,s_ok);
    constant s1:STD_LOGIC_VECTOR(2 downto 0):="000";
    constant s2:STD_LOGIC_VECTOR(2 downto 0):="001";
    constant s3:STD_LOGIC_VECTOR(2 downto 0):="010";
    constant s4:STD_LOGIC_VECTOR(2 downto 0):="011";
    constant s_ok:STD_LOGIC_VECTOR(2 downto 0):="100";
    signal state_cur,state_next:STD_LOGIC_VECTOR(2 downto 0);
begin
    process(clk,rst_n)
    begin
        if rst_n='0' then
            state_cur<=s1;
        elsif clk'event and clk='1' then
            state_cur<=state_next;
        end if;
    end process;

    process(state_cur,data)
    begin
        case state_cur is
            when s1=>
                if data='0' then
                    state_next<=s2;
                else
                     state_next<=s1;
```

```
                end if;
            when s2=>
                if data='1' then
                    state_next<=s3;
                else
                    state_next<=s1;
                end if;
            when s3=>
                if data='1' then
                    state_next<=s4;
                else
                    state_next<=s1;
                end if;
            when s4=>
                if data='0' then
                    state_next<=s_ok;
                else
                    state_next<=s1;
                end if;
            when s_ok=>
                    state_next<=s1;
            when others=>
                    state_next<=s1;
        end case;
    end process;

    check<='1' when state_cur=s_ok else
        '0';

end Behavioral;
```

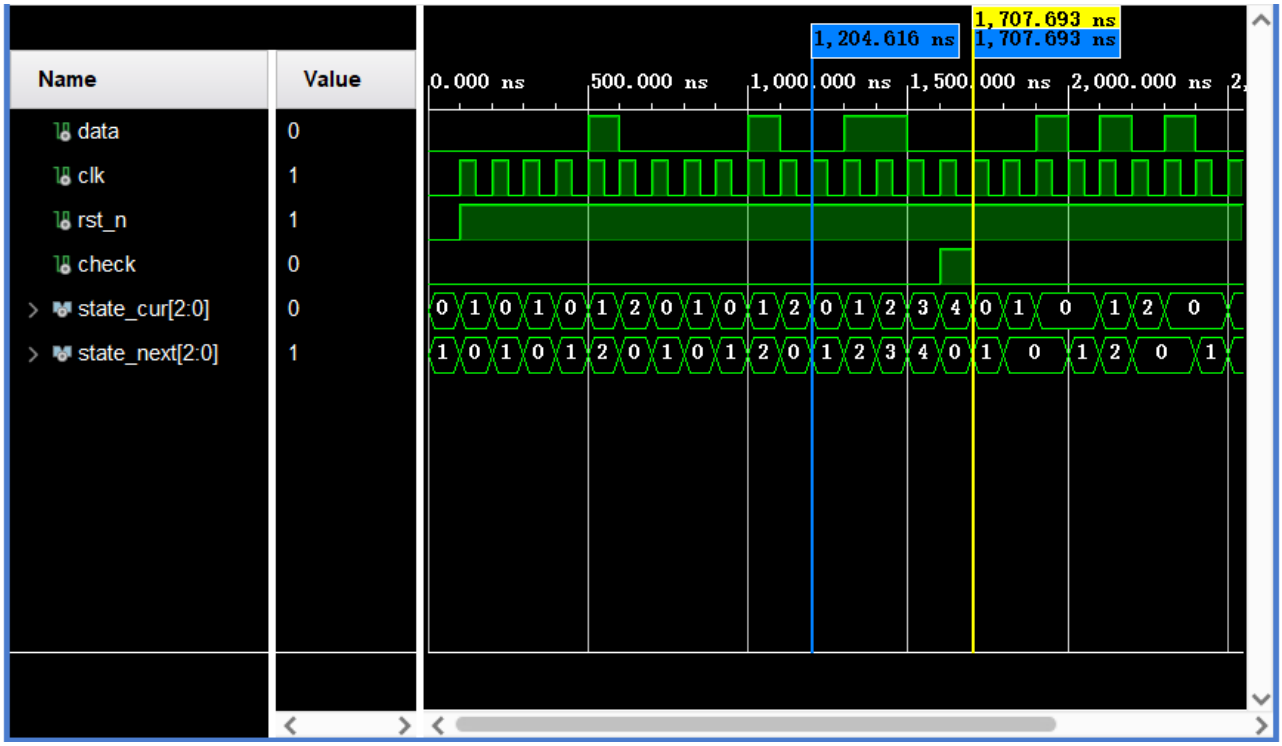
Testbench code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity test_sequ is
--  Port ( );
end test_sequ;

architecture Behavioral of test_sequ is
    component sequ_scan is
        Port ( data : in STD_LOGIC;
            clk : in STD_LOGIC;
```

```
            rst_n : in STD_LOGIC;
            check : out STD_LOGIC);
    end component;
    signal data : STD_LOGIC;
    signal clk : STD_LOGIC;
    signal rst_n : STD_LOGIC;
    signal check : STD_LOGIC;
begin
    uut:sequ_scan port map(
        data=>data,
        clk=>clk,
        rst_n=>rst_n,
        check=>check
    );
    process
        variable num:STD_LOGIC_VECTOR(3 downto 0);
    begin
        rst_n<='0';
        data<='0';
        wait for 100 ns;
        rst_n<='1';
        for i in 0 to 15 loop
            num:=STD_LOGIC_VECTOR(to_unsigned(i,4));
            data<=num(0);
            wait for 100 ns;
            data<=num(1);
            wait for 100 ns;
            data<=num(2);
            wait for 100 ns;
            data<=num(3);
            wait for 100 ns;
        end loop;
    end process;

    process
    begin
        clk<='0';
        wait for 100 ns;
        while true loop
            clk<='1';
            wait for 50 ns;
            clk<='0';
            wait for 50 ns;
        end loop;
```

```
    end process;

end Behavioral;
```

Moore--Simulation waveform



Mealy 机

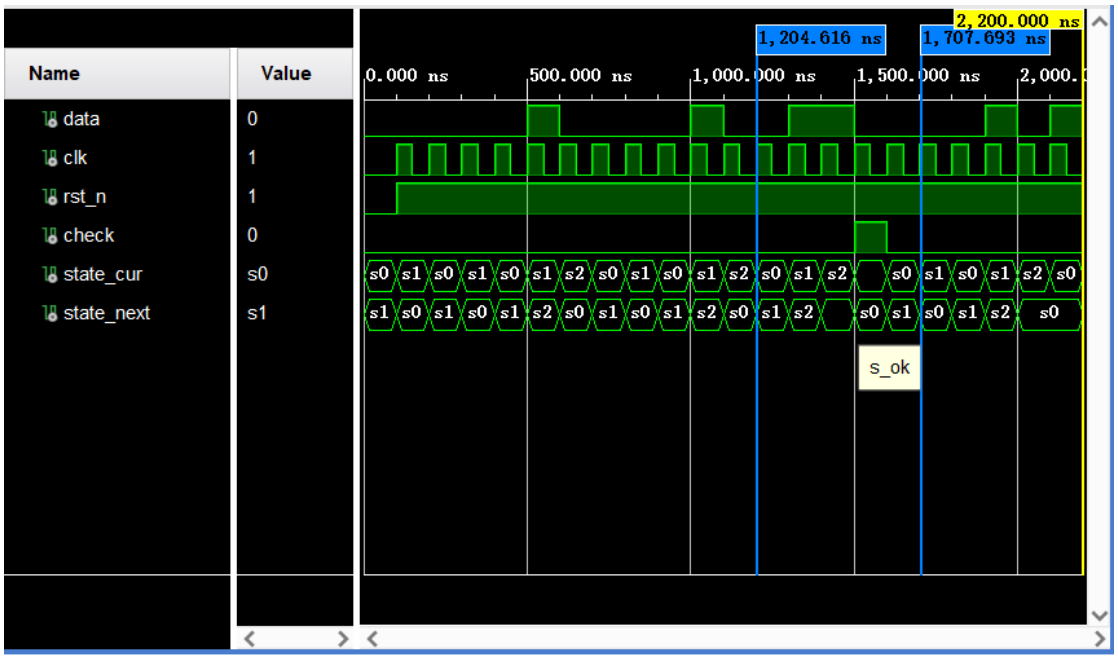
Mealy--Source code

```
--Mealy Machine
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sequ_scan is
    Port ( data : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst_n : in STD_LOGIC;
           check : out STD_LOGIC);
end sequ_scan;

architecture Behavioral of sequ_scan is
    type state_enum is (s0,s1,s2,s_ok);
    signal state_cur,state_next:state_enum;
```

```
begin
    process(clk,rst_n)
    begin
        if rst_n='0' then
            state_cur<=s0;
        elsif clk'event and clk='1' then
            state_cur<=state_next;
        end if;
    end process;

    process(state_cur,data)
    begin
        case state_cur is
            when s0=>
                check<='0';
                if data='0' then
                    state_next<=s1;
                else
                     state_next<=s0;
                end if;
            when s1=>
                check<='0';
                if data='1' then
                    state_next<=s2;
                else
                    state_next<=s0;
                end if;
            when s2=>
                check<='0';
                if data='1' then
                    state_next<=s_ok;
                else
                    state_next<=s0;
                end if;
            when s_ok=>
                if data='0' then
                    check<='1';
                else
                    check<='0';
                end if;
                state_next<=s0;
            when others=>
                check<='0';
                state_next<=s0;
```

```
        end case;
    end process;


end Behavioral;
```

Mealy--Simulation waveform



HOMEWORK7

Source code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity light_dynamic is
    Port ( clk : in STD_LOGIC;
           rst_n : in STD_LOGIC;
           -- light_en : in STD_LOGIC;
           mode : in STD_LOGIC_VECTOR(1 downto 0);
           light : out STD_LOGIC_VECTOR (7 downto 0));
end light_dynamic;

architecture Behavioral of light_dynamic is
    signal count_div:STD_LOGIC_VECTOR(23 downto 0);
    signal clk_2hz:STD_LOGIC;
    signal count_light:STD_LOGIC_VECTOR(4 downto 0);
    signal count_light_start:STD_LOGIC_VECTOR(4 downto 0);
    signal count_light_stop:STD_LOGIC_VECTOR(4 downto 0);
    signal count_light_en:STD_LOGIC;
    signal count_light_rst_n:STD_LOGIC;
    signal count_light_work:STD_LOGIC;

    constant ready:STD_LOGIC_VECTOR(1 downto 0):="00";
    constant step1:STD_LOGIC_VECTOR(1 downto 0):="01";
    constant step2:STD_LOGIC_VECTOR(1 downto 0):="10";
    constant step3:STD_LOGIC_VECTOR(1 downto 0):="11";
    --constant step4:STD_LOGIC_VECTOR(2 downto 0):="100";
    signal state_cur,state_next:STD_LOGIC_VECTOR(1 downto 0);
begin
    -- divide 50MHz-->2Hz,25M dividion,12.5M counters
    -- 12,500,000d=1011 1110 1011 1100 0010 0000b
    -- 12,499,999d=1011 1110 1011 1100 0001 1111b

    --  process(clk,rst_n)
    --  begin
    --      if rst_n='0' then
    --          count_div<="000000000000000000000000";
    --          clk_2hz<='0';
    --      elsif clk'event and clk='1' then
    --          if count_div="101111101011110000011111" then
    --              count_div<="000000000000000000000000";
    --              clk_2hz<=not clk_2hz;
```

```
--          else
--              count_div<=count_div+1;
--          end if;
--      end if;
--  end process;
clk_2hz<=clk;

-- state machine
process(clk,rst_n)
begin
    if rst_n='0' then
        state_cur<=ready;
    elsif clk'event and clk='1' then
        state_cur<=state_next;
    end if;
end process;

process(state_cur,mode,count_light_work)
begin
    case state_cur is
        when ready=>
            case mode is
                when "00"=> state_next<=ready;
                when "01"=> state_next<=step1;
                when "10"=> state_next<=step2;
                when "11"=> state_next<=step3;
                when others=> state_next<=ready;
            end case;
        when others=>
            if count_light_work='0' then
                state_next<=ready;
            else
                state_next<=state_next;
            end if;
    end case;
end process;

process(state_cur)
begin
    case state_cur is
        when ready=>
            count_light_rst_n<='0';
            count_light_en<='0';
        when others=>
```

```
                count_light_rst_n<='1';
                count_light_en<='1';
            end case;
        end process;

        -- counter's start & end decoder

        process(state_cur,mode,rst_n)
        begin
            if state_cur=ready then
                if rst_n='0' then
                    count_light_start<="00000";
                else
                    case mode is
                        when "00"=> count_light_start<="00000";
                        when "01"=> count_light_start<="00001";
                        when "10"=> count_light_start<="00011";
                        when "11"=> count_light_start<="01011";
                        when others=> count_light_start<="00000";
                    end case;
                end if;
            end if;
        end process;
        process(state_cur,mode)
        begin
            if state_cur=ready then
                case mode is
                    when "00"=> count_light_stop<="00000";
                    when "01"=> count_light_stop<="00010";
                    when "10"=> count_light_stop<="01010";
                    when "11"=> count_light_stop<="10100";
                    when others=> count_light_stop<="00000";
                end case;
            end if;
        end process;

        -- light counter
        -- step 1: 3 counters (0_0000~0_0010)
        -- step 2: 8 counters (0_0011~0_1010)
        -- step 3: 5 counters (0_1011~0_1111)
        -- step 4: 5 counters (1_0000~1_0100)
        process(clk_2hz,count_light_en,count_light_rst_n,count_light_start,
    count_light_stop)
        begin
```

```
    if count_light_rst_n='0' then
        count_light<=count_light_start;
        count_light_work<='1';
    elsif clk_2hz'event and clk_2hz='1' then
        if count_light_en='1' then
            if count_light=count_light_stop then
                count_light_work<='0';
            else
                count_light<=count_light+1;
            end if;
        end if;
    end if;
end process;

-- light decoder
process(count_light)
begin
    case count_light is
        -- step 1
        when "00000"|"00010"=>
            light<=(others=>'0');
        when "00001"=>
            light<=(others=>'1');

        -- step 2
        when "00011"=>
            light<=(7=>'1',others=>'0');
        when "00100"=>
            light<=(6=>'1',others=>'0');
        when "00101"=>
            light<=(5=>'1',others=>'0');
        when "00110"=>
            light<=(4=>'1',others=>'0');
        when "00111"=>
            light<=(3=>'1',others=>'0');
        when "01000"=>
            light<=(2=>'1',others=>'0');
        when "01001"=>
            light<=(1=>'1',others=>'0');
        when "01010"=>
            light<=(0=>'1',others=>'0');

        -- step 3
        when "01011"=>
```

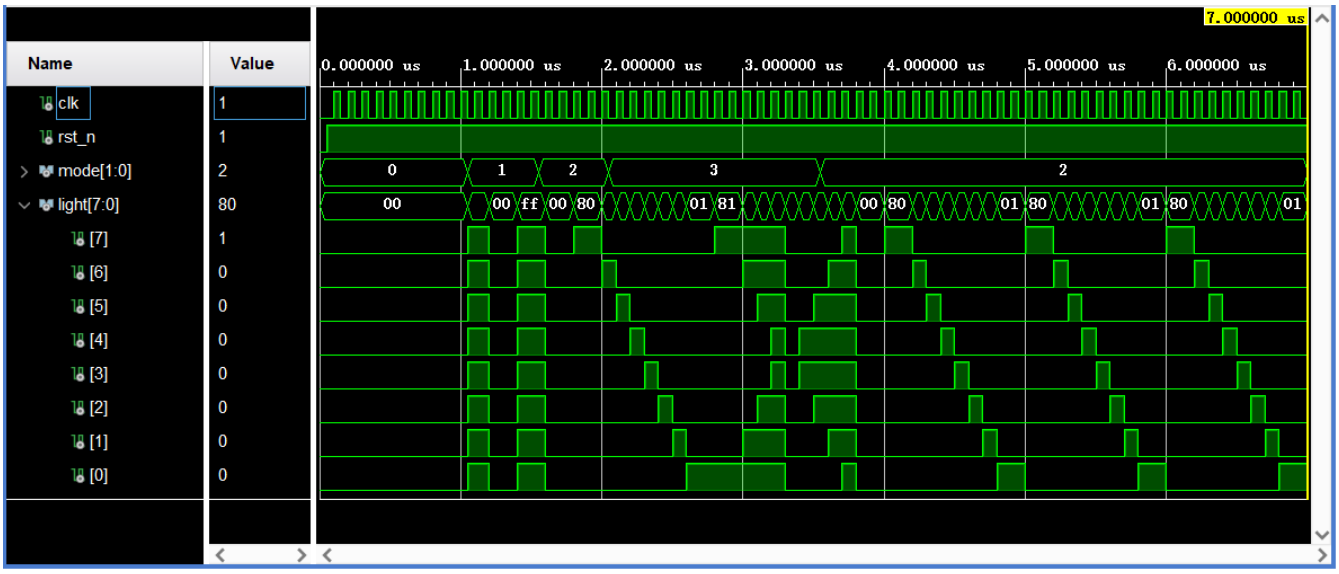
```
                light<=(7=>'1',0=>'1',others=>'0');
            when "01100"=>
                light<=(7 downto 6=>'1',1 downto 0=>'1',others=>'0');
            when "01101"=>
                light<=(7 downto 5=>'1',2 downto 0=>'1',others=>'0');
            when "01110"=>
                light<=(others=>'1');
            when "01111"=>
                light<=(others=>'0');

            -- step 4
            when "10000"=>
                light<=(4 downto 3=>'1',others=>'0');
            when "10001"=>
                light<=(5 downto 2=>'1',others=>'0');
            when "10010"=>
                light<=(6 downto 1=>'1',others=>'0');
            when "10011"=>
                light<=(others=>'1');
            when "10100"=>
                light<=(others=>'0');

            when others=>light<=(others=>'0');
        end case;
    end process;

end Behavioral;
```

Testbench code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity test_light is
--  Port ( );
end test_light;

architecture Behavioral of test_light is
    component light_dynamic is
        Port ( clk : in STD_LOGIC;
            rst_n : in STD_LOGIC;
            mode : in STD_LOGIC_VECTOR(1 downto 0);
            light : out STD_LOGIC_VECTOR (7 downto 0));
    end component;
    signal clk : STD_LOGIC;
```

```
    signal rst_n : STD_LOGIC;
    signal mode : STD_LOGIC_VECTOR(1 downto 0);
    signal light : STD_LOGIC_VECTOR (7 downto 0);
begin
    uut:light_dynamic port map(
        clk=>clk,
        rst_n=>rst_n,
        mode=>mode,
        light=>light
    );
    process
        variable num:STD_LOGIC_VECTOR(3 downto 0);
    begin
        rst_n<='0';
        mode<="00";
        wait for 50 ns;
        rst_n<='1';
        wait for 1000 ns;
        mode<="01"; wait for 500 ns;
        mode<="10"; wait for 500 ns;
        mode<="11"; wait for 1500 ns;
        mode<="10"; wait;
    end process;

    process
    begin
        clk<='0';
        wait for 100 ns;
        while true loop
            clk<='1';
            wait for 50 ns;
            clk<='0';
            wait for 50 ns;
        end loop;
    end process;

end Behavioral;
```

Simulation waveform


Name
Value
clk
1
rst_n
1
mode[1:0]
2
light[7:0]
80
[7]
1
[6]
0
[5]
0
[4]
0
[3]
0
[2]
0
[1]
0
[0]
0
7.000000 us
0.000000 us
1.000000 us
2.000000 us
3.000000 us
4.000000 us
5.000000 us
6.000000 us
0
1
2
3
2
00
00
ff
00
80
01
81
00
80
01
80
01
80
01

硬件测试还有视频