DESIGN REPORT

I. DESIGN IDEA

1. MODEL DIVIDE

First build a component model for a single cell (***SINGEL_CELL***)， and than connect cell models in a top file (***RUN_MODEL***)。There are two source file in all. And there are two testbench for each model.

Use vivado to do the simulation.

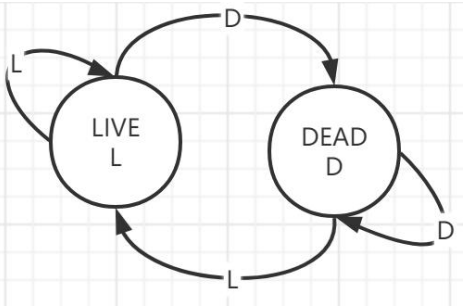
2. SINGLE_CELL

SINGLE_CELL is the model for a single cell。

The ports are as follows, as same as that in the request document.

Name	Direction	Description
clk	Input	Global clock
srstn	Input	Global synchronous active low reset (force to 0: dead)
ce	Input	Clock enable
load	Input	When set, take W as next state
NW	Input	Current state of the north-west neighbour (0: dead, 1: live)
N	Input	Current state of the north neighbour (0: dead, 1: live)
NE	Input	Current state of the north-east neighbour (0: dead, 1: live)
E	Input	Current state of the east neighbour (0: dead, 1: live)
SE	Input	Current state of the south-east neighbour (0: dead, 1: live)
S	Input	Current state of the south neighbour (0: dead, 1: live)
SW	Input	Current state of the south-west neighbour (0: dead, 1: live)
W	Input	Current state of the west neighbour (0: dead, 1: live)
state	Output	Current state of CELL (0: dead, 1: live)

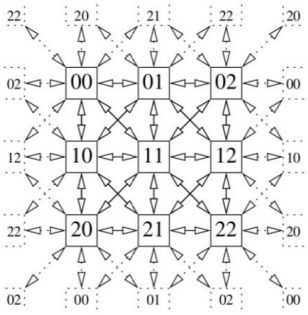
The single cell is designed as a Moore FSM，the state change is as follows：



The output ***state*** is directly determined by the current state, and the next state is determined by input signals and current state.

3. RUN_NET

This module is the top model which connect cells together according to the relationship shows in the picture below:



There is some relationship between a continuous storage index and the position of the cells, which can be easily calculated by express the row and column respectively.

And the statement in this architecture is better to be Synchronous. If the signal between cells are connected correctly , the cells component will change the state itself.

II. PROGRAMMING EXPLAINATION

1. SINGLE_CELL.vhd

Define ***ENTITY SINGLE_CELL*** , and the ***PORT*** as same as requested in the document。

```
ENTITY SINGLE_CELL IS
PORT (clk,srstn,ce,load,NW,N,NE,W,E,SW,S,SE:IN STD_LOGIC;
      state:OUT STD_LOGIC );
END SINGLE_CELL;
```

Define ***ARCHITECTURE Behavioral***. Signal

current_state : refers to the current state of the machine,

next_state : refers to the next state of the machine.

Live_num : refers to the live cells around the cell in all.

```
ARCHITECTURE Behavioral OF SINGLE_CELL IS

SIGNAL current_state:STD_LOGIC;
SIGNAL next_state:STD_LOGIC;
SIGNAL live_num:INTEGER;
```

Define ***FNUCTION convert_logic*** to convert ***STD_LOGIC_VECTOR*** into ***INTEGER,*** in order to make it easier to calculate the alive cell around.

```
FUNCTION convert_logic(a:STD_LOGIC)RETURN INTEGER IS
    BEGIN
        IF a='1' THEN
            RETURN 1;
        ELSE
            RETURN 0;
        END IF;
END FUNCTION convert_logic;
```

Calculate the live_num after begin of the architecture.

```
live_num<=((convert_logic(NW)+convert_logic(N))+(convert_logic(NE)+convert_logic(E)))
        +((convert_logic(SE)+convert_logic(S))+(convert_logic(SW)+convert_logic(W)));
```

Use 3 process to manage the FSM. The three Processes are ***REG, STATE_OUT*** and ***NEXT_GET***.

PROCESS REG: the timing process, determines what to do when the ***clk*** rising edge arrive. Use Synchronous reset. ***srstn***(reset) has higher priority than ***ce*** (clk enable).

```
REG:PROCESS(clk,srstn)
BEGIN
    IF clk'EVENT AND clk='1' THEN
        IF srstn='0' THEN
            current_state<='0';
        ELSE
            IF ce='1' THEN
                current_state<=next_state;
            END IF;
        END IF;
    END IF;
END PROCESS REG;
```

STATE_OUT: Output the current state when current state changes

```
STATE_OUT:PROCESS(current_state)
BEGIN
    state<=current_state;
END PROCESS STATE_OUT;
```

NEXT_GET: Determine the next state in this FSM. When the cell is live and ***live_num***=2 or ***live_num***=3 , the cell remain live else the cell will become dead. When the cell is dead and ***live_num***=3, the cell will resurrection. ***Load*** has a higher priority, if load is set the next stet will become ***W*** anyhow.

```
NEXT_GET:PROCESS(load,live_num)
BEGIN
    IF load='1'THEN
        next_state<=W;
    ELSE
        IF current_state='1' THEN
            IF live_num=2 THEN
                next_state<='1';
            ELSIF live_num=3 THEN
                next_state<='1';
            ELSE
                next_state<='0';
            END IF;
        ELSIF current_state='0' THEN
            IF live_num=3 THEN
                next_state<='1';
            ELSE
                next_state<='0';
            END IF;
        ELSE
                next_state<=current_state;
        END IF;
    END IF;
END PROCESS NEXT_GET;
```

2. RUN_NET.vhd

Define ***ENTIRY RUN_NET***, Define GENERIC INTEGER r and c.

r/c : the row and column number of the net.

left_side : the input state which can be load in ***W.***

right_side : the output column of the net.

read_cell : a port which is in convenience of debug, shows the state of each cell in the net.

```
ENTITY RUN_NET IS
GENERIC(r:INTEGER:=8;c:INTEGER:=8);
PORT (clk,srstn,ce,load,pre_load:IN STD_LOGIC;
      left_side:IN STD_LOGIC_VECTOR(r-1 DOWNTO 0);
      right_side:OUT STD_LOGIC_VECTOR(r-1 DOWNTO 0);
      read_cell:OUT STD_LOGIC_VECTOR(r*c-1 DOWNTO 0)
       );
END RUN_NET;
```

Define the Component in order to instantiate the cell model. Define vectors which range is r*c to store the state of cells and the states of the cells around.

```
ARCHITECTURE Behavioral of RUN_NET IS

COMPONENT SINGLE_CELL
PORT (clk,srstn,ce,load,NW,N,NE,W,E,SW,S,SE:IN STD_LOGIC;
      state:OUT STD_LOGIC );
END COMPONENT;
SIGNAL NW,N,NE,W,E,SW,S,SE,cell:STD_LOGIC_VECTOR(R*C-1 DOWNTO 0);
```

Use the ***WITH/SELECT*** statement to determine whether to load the state from port ***left_side***. If the load is set, then connect ***W*** of the first column with ***left_side***, if load is not set, connect ***W*** of the first column with the state of the cell of the last column.

```
WIN: FOR i IN 0 TO r-1 GENERATE
        WITH pre_load SELECT
        W(i*c)<=  left_side(i)  WHEN '1' ,
                  cell((i+1)*c-1) WHEN '0' ,
                  'X'WHEN OTHERS;
END GENERATE WIN;
```

Then use parallel statement to connect the cells net. Divide the connection relationship into 3 parts: the internal part, the edge, and the corner. (The row and column start from 0)

First connect the cells of the internal part of the net (from row 1 to r-2 and column 1 to c-2)

```
---------internal part-----------
    CONC1:FOR i IN 1 TO r-2 GENERATE
    BEGIN
        CONC2:FOR j IN 1 TO c-2 GENERATE
        BEGIN
            NW(i*c+j)<=cell((i-1)*c+j-1);
            N(i*c+j)<=cell((i-1)*c+j);
            NE(i*c+j)<=cell((i-1)*c+j+1);
            W(i*c+j)<=cell(i*c+j-1);
            E(i*c+j)<=cell(i*c+j+1);
            SW(i*c+j)<=cell((i+1)*c+j-1);
            S(i*c+j)<=cell((i+1)*c+j);
            SE(i*c+j)<=cell((i+1)*c+j+1);
        END GENERATE CONC2;
    END GENERATE CONC1;
```

Second, connect cells on the edge but do not contain the cells on the corner.The Left Edge is cells on the column 0 from row 1 to row r-2:

```
-------------left edge-----------------------------
    LEFT:FOR i IN 1 TO r-2 GENERATE
    BEGIN
            NW(i*c)<=cell(i*c-1);
            N(i*c)<=cell((i-1)*c);
            NE(i*c)<=cell((i-1)*c+1);
        --      W(i*c)<=cell(i*c+c-1);
            E(i*c)<=cell(i*c+1);
            SW(i*c)<=cell((i+2)*c-1);
            S(i*c)<=cell((i+1)*c);
            SE(i*c)<=cell((i+1)*c+1);
    END GENERATE LEFT;
```

The right edge is the cells form 1 to r-2 on the column c-1:

```
-----------------right edge
    RIGHT:FOR i IN 2 TO r-1 GENERATE
        NW(i*c-1)<=cell((i-1)*c-2);
        N(i*c-1)<=cell((i-1)*c-1);
        NE(i*c-1)<=cell((i-2)*c);
        W(i*c-1)<=cell(i*c-2);
        E(i*c-1)<=cell((i-1)*c);
        SW(i*c-1)<=cell((i+1)*c-2);
        S(i*c-1)<=cell((i+1)*c-1);
        SE(i*c-1)<=cell(i*c);
    END GENERATE RIGHT;
```

The up side is cells from 1 to c-2 on the row 0:

```
----------------up edge---------------------
    UP:FOR i IN 1 TO c-2 GENERATE
        NW(i)<=cell((r-1)*c+i-1);
        N(i)<=cell((r-1)*c+i);
        NE(i)<=cell((r-1)*c+i+1);
        W(i)<=cell(i-1);
        E(i)<=cell(i+1);
        SW(i)<=cell(c+i-1);
        S(i)<=cell(c+i);
        SE(i)<=cell(c+i+1);
    END GENERATE UP;
```

The down side is cells from 1 to c-2 on the row r-1:

```
---------------down edge--------------------
    DOWN:FOR i IN 1 TO c-2 GENERATE
        NW((r-1)*c+i)<=cell((r-2)*c+i-1);
        N((r-1)*c+i)<=cell((r-2)*c+i);
        NE((r-1)*c+i)<=cell((r-2)*c+i+1);
        W((r-1)*c+i)<=cell((r-1)*c+i-1);
        E((r-1)*c+i)<=cell((r-1)*c+i+1);
        SW((r-1)*c+i)<=cell(i-1);
        S((r-1)*c+i)<=cell(i);
        SE((r-1)*c+i)<=cell(i+1);
    END GENERATE DOWN;
```

The third part is the connection relationships of four corners.

The north west corner is cell on row 0 column 0:

```
----------------north west--------------
NWCORNER: BLOCK
BEGIN
    NW(0)<=cell(c*r-1);
    N(0)<=cell((r-1)*c);
    NE(0)<=cell((r-1)*c+1);
--  W(0)<=cell(c-1);
    E(0)<=cell(1);
    SW(0)<=cell(c*2-1);
    S(0)<=cell(c);
    SE(0)<=cell(c+1);
END BLOCK NWCORNER;
```

The north east corner is the cell on row 0 column c-1:

```
-----------------north east-----------------
NECORNER: BLOCK
BEGIN
    NW(c-1)<=cell(r*c-2);
    N(c-1)<=cell(r*c-1);
    NE(c-1)<=cell((r-1)*c);
    W(c-1)<=cell(c-2);
    E(c-1)<=cell(0);
    SW(c-1)<=cell(2*c-2);
    S(c-1)<=cell(2*c-1);
    SE(c-1)<=cell(c);
END BLOCK NECORNER;
SWCORNER: BLOCK
```

The south west corner is the cell on row r-1 column 0:

```
-------------------------south west--------------------------
SWCORNER: BLOCK
BEGIN
    NW(c*(r-1))<=cell((c-1)*r-1);
    N(c*(r-1))<=cell((c-2)*r);
    NE(c*(r-1))<=cell((c-2)*r+1);
--W(c*(r-1))<=cell(c*r-1);
    E(c*(r-1))<=cell((c-1)*r+1);
    SW(c*(r-1))<=cell(c-1);
    S(c*(r-1))<=cell(0);
    SE(c*(r-1))<=cell(1);
END BLOCK SWCORNER;
```

The south east corner is the cell on the row r-1 column c-1:

```
-------------------------------south east-------------------------------
SECORNER: BLOCK
BEGIN
    NW(c*r-1)<=cell((r-1)*c-2);
    N(c*r-1)<=cell((r-1)*c-1);
    NE(c*r-1)<=cell((r-2)*c);
    W(c*r-1)<=cell(r*c-2);
    E(c*r-1)<=cell((r-1)*c);
    SW(c*r-1)<=cell(c-2);
    S(c*r-1)<=cell(c-1);
    SE(c*r-1)<=cell(0);
END BLOCK SECORNER;
```

The index relationship of a cell and the cells around it is easy to calculate.

Then generate the ***SINGLE_CELL*** components.

```
GENERATE_CELL1:FOR i IN 0 TO R-1 GENERATE
BEGIN
    GENERATE_CELL2:FOR j IN 0 TO C-1 GENERATE
        CELL_INTIA:SINGLE_CELL PORT MAP(clk=>clk,srstn=>srstn,ce=>ce,load=>load,NW=>NW(i*C+j),N=>N(i*C+j),
            NE=>NE(i*C+j),W=>W(i*C+j),E=>E(i*C+j),SW=>SW(i*C+j),S=>S(i*C+j),SE=>SE(i*C+j),state=>cell(i*C+j));
    END GENERATE GENERATE_CELL2;
END GENERATE GENERATE_CELL1;
```

And output the state of cells on column c-1 by ***right_side.***

```
OUTPUT:FOR i IN 0 TO r-1 generate
    right_side(i)<=cell(i*c+c-1);
END GENERATE OUTPUT;
read_cell<=cell;
END Behavioral;
```

3. RUN_SIM.vhd

Define the signals to connect with ports of components

HPCLK: time length of a half clock period

clk : clock

srstn : reset signal

ce : clock enable

load : determine whether to load state from W

pre_load : a test signal in debug

left_side : the input state to load to W

right_side : the output state from the rightest column

read_cell : show the state of each cell.

```
ENTITY RUN_SIM IS
GENERIC(r:integer:=8;c:integer:=8);
END RUN_SIM;

ARCHITECTURE Behavioral OF RUN_SIM IS
COMPONENT RUN_NET
GENERIC(r:INTEGER:=8;c:INTEGER:=8);
Port (clk,srstn,ce,load,pre_load:IN STD_LOGIC;
      right_side:OUT STD_LOGIC_VECTOR(r-1 downto 0);
      left_side:IN STD_LOGIC_VECTOR(r-1 downto 0);
      read_cell :OUT  STD_LOGIC_VECTOR(r*c-1 downto 0)
      );
END COMPONENT;

CONSTANT HPCLK:TIME:=10ns;
SIGNAL clk:STD_LOGIC:='0';
SIGNAL srstn:STD_LOGIC:='1';
SIGNAL ce:STD_LOGIC:='1';
SIGNAL load:STD_LOGIC:='0';
SIGNAL pre_load:STD_LOGIC:='0';

SIGNAL left_side:STD_LOGIC_VECTOR(r-1 DOWNTO 0);
SIGNAL right_side:STD_LOGIC_VECTOR(r-1 DOWNTO 0);
SIGNAL read_cell :STD_LOGIC_VECTOR(c*r-1 DOWNTO 0);
```

Use the clock with period of 20ns

```
GENERATE_CLK:process
begin
    clk<='0';
    wait for HPCLK;
    clk<='1';
    wait for HPCLK;
END PROCESS GENERATE_CLK;
```

Synchronous reset

```
SIM:PROCESS
BEGIN

    WAIT FOR 2*HPCLK;
    srstn<='1';
    WAIT FOR 2*HPCLK;
    srstn<='0';
    WAIT FOR 2*HPCLK;
    srstn<='1';
    ce<=1;
```

Load different ***left_side*** to simulate.

```
    left_side<="11011111";
    pre_load<='1';
    load<='1';
    WAIT FOR 2*HPCLK;
    load<='0';
    pre_load<='0';
    WAIT FOR 10*HPCLK;

    left_side<="11111111";
    pre_load<='1';
    load<='1';
    WAIT FOR 2*HPCLK;
    load<='0';
    pre_load<='0';
    WAIT FOR 10*HPCLK;

    left_side<="11100111";
    pre_load<='1';
    load<='1';
    WAIT FOR 2*HPCLK;
    load<='0';
    pre_load<='0';
    WAIT FOR 10*HPCLK;


END PROCESS SIM;

RUN_NET_INTIA:RUN_NET PORT MAP(clk=>clk,srstn=>srstn,ce=>ce,load=>load,pre_load=>pre_load,
        right_side=>right_side,left_side=>left_side,read_cell=>read_cell);

end Behavioral;
```

4. SINGLE_CELL_SIM.vhd

Define the signals to connect with ports of components, and design signals according to the table in the request document.

```
ARCHITECTURE Behavioral OF SINGLE_CELL_SIM IS
COMPONENT SINGLE_CELL
PORT (clk,srstn,ce,load,NW,N,NE,W,E,SW,S,SE:IN STD_LOGIC;
      state:OUT STD_LOGIC);
END COMPONENT;

CONSTANT HPCLK:TIME:=10ns;
SIGNAL clk:STD_LOGIC;
SIGNAL srstn:STD_LOGIC;
SIGNAL ce:STD_LOGIC;
SIGNAL load:STD_LOGIC;
SIGNAL NW,N,NE,W,E,SW,S,SE:STD_LOGIC;
SIGNAL state:STD_LOGIC;
```

Generate clk:

```
GENERATE_CLK:PROCESS
BEGIN
    clk<='0';
    WAIT FOR HPCLK;
    clk<='1';
   WAIT FOR HPCLK;
END PROCESS GENERATE_CLK ;
```

Then Synchronous reset:

```
BEGIN
SIM_CELL:PROCESS
BEGIN
    ce<='1';
    srstn<='0';
    wait for 2*HPCLK;
    srstn<='1';
    wait for 2*HPCLK;
    srstn<='0';
```

Load the test data as follows:

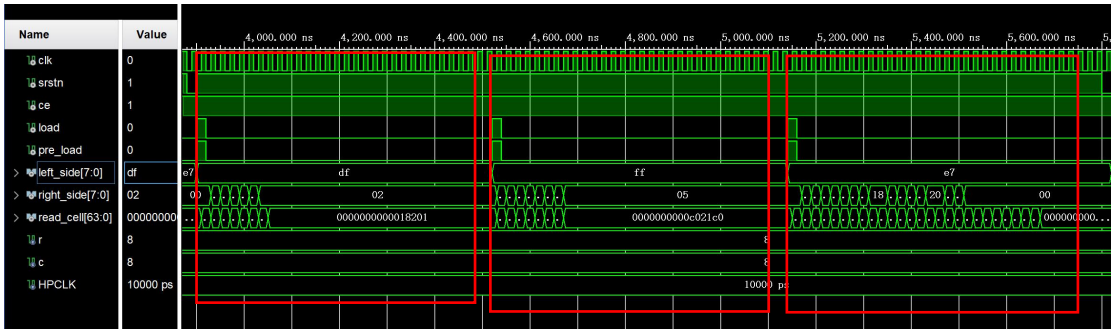
```
    WAIT FOR 2*HPCLK;
    load<='1';
    WAIT FOR 2*HPCLK;
    load<='0';


    WAIT FOR 6*HPCLK;
    NW<='1';
    N<='0';
    NE<='0';
    W<='1';
    E<='1';
    SW<='0';
    S<='0';
    SE<='0';
    WAIT FOR 6*HPCLK;
END PROCESS SIM_CELL;
```

Instantiate ***SINGLE_CELL:***

```
CELL_INTIA:SINGLE_CELL PORT MAP(clk=>clk,srstn=>srstn,ce=>ce,load=>load,NW=>NW,
                        N=>N,NE=>NE,W=>W,E=>E,SW=>SW,S=>S,SE=>SE,state=>state);
```

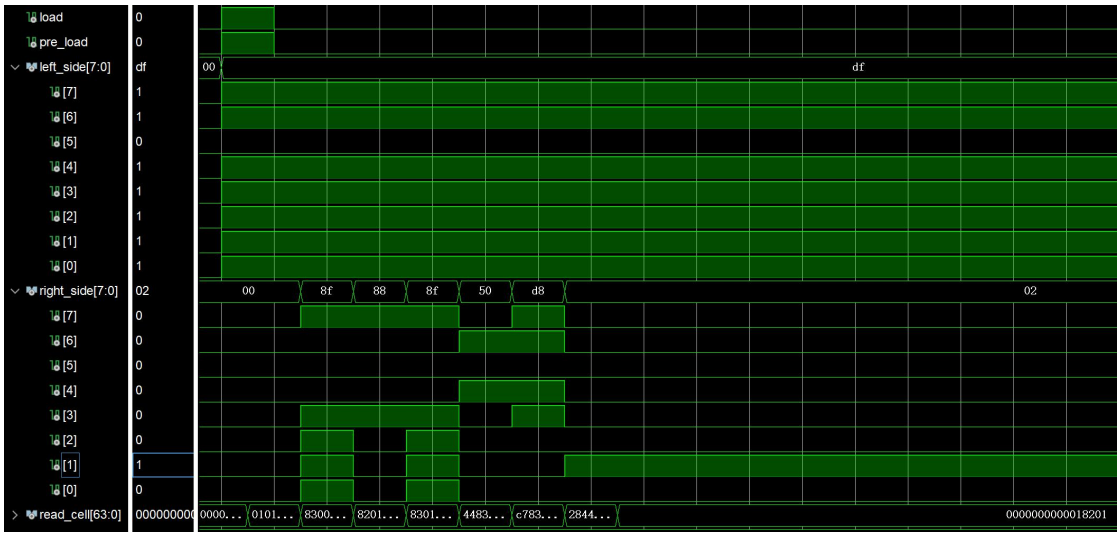
III. SIMULAITON WAVE

1. RUN_SIM result:

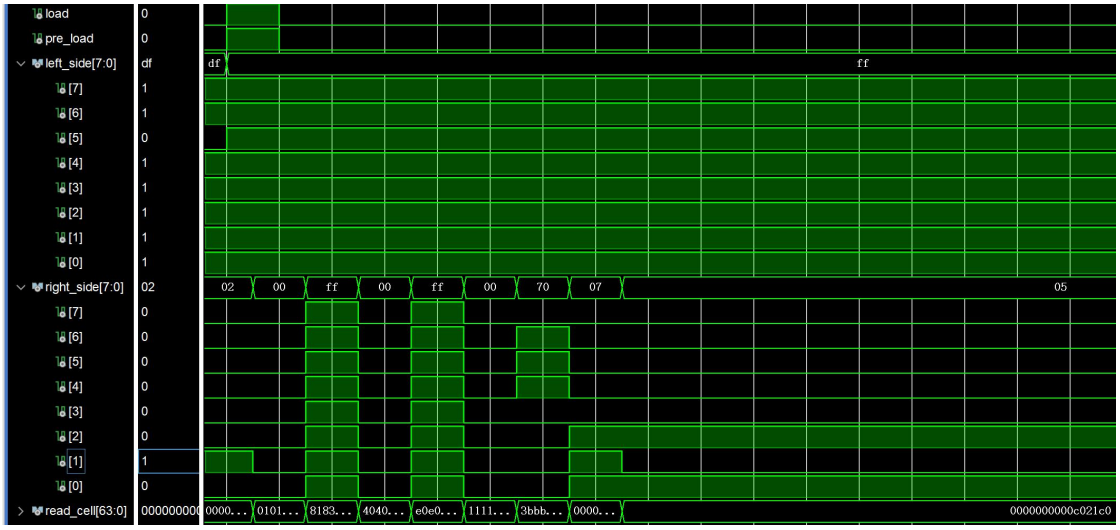


Load 3 times of a 8*8 net, and compare the result with a 8*8 net on the internet

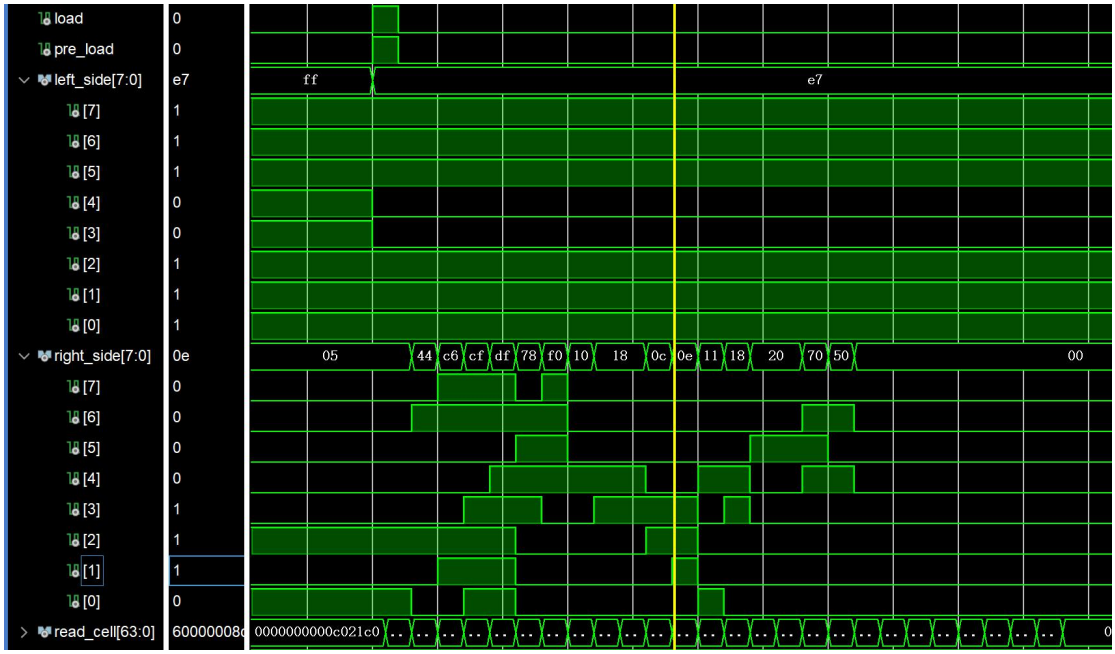
First time:



Second time:

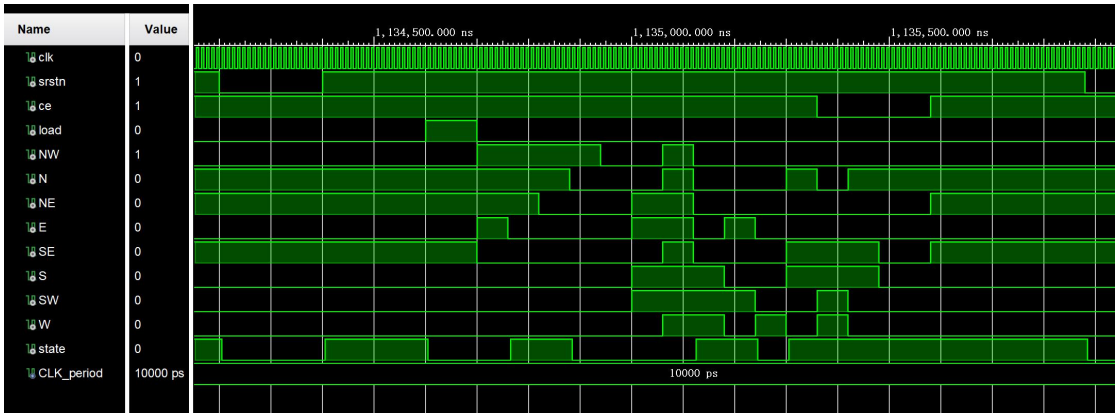


Third time:



2. SINGLE_CELL_SIM result

The ***ce, srstn, state*** can be test in this simulation:



APPENDIX: SOURCE AND SIMULATION CODINGS

1. SINGLE_CELL.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
ENTITY SINGLE_CELL IS
PORT (clk,srstn,ce,load,NW,N,NE,W,E,SW,S,SE:IN STD_LOGIC;
       state:OUT STD_LOGIC );
END SINGLE_CELL;
ARCHITECTURE Behavioral OF SINGLE_CELL IS
SIGNAL current_state:STD_LOGIC;
SIGNAL next_state:STD_LOGIC;
SIGNAL live_num:INTEGER;
FUNCTION convert_logic(a:STD_LOGIC)RETURN INTEGER IS
    BEGIN
        IF a='1' THEN
            RETURN 1;
        ELSE
            RETURN 0;
        END IF;
END FUNCTION convert_logic;
BEGIN
REG:PROCESS(clk,srstn)
BEGIN
    IF clk'EVENT AND clk='1' THEN
        IF srstn='0' THEN
            current_state<='0';
        ELSE
            IF ce='1' THEN
                current_state<=next_state;
            END IF;
        END IF;
    END IF;
END PROCESS REG;
STATE_OUT:PROCESS(current_state)
BEGIN
    state<=current_state;
END PROCESS STATE_OUT;

live_num<=((convert_logic(NW)+convert_logic(N))+(convert_logic(NE)+convert_logic(E)))
        +((convert_logic(SE)+convert_logic(S))+(convert_logic(SW)+convert_logic(W)));
NEXT_GET:PROCESS(load,live_num)
BEGIN
    IF load='1'THEN
        next_state<=W;
    ELSE
        IF current_state='1' THEN
            IF live_num=2 THEN
                next_state<='1';
            ELSIF live_num=3 THEN
                next_state<='1';
            ELSE
                next_state<='0';
            END IF;
        ELSIF current_state='0' THEN
            IF live_num=3 THEN
                next_state<='1';
            ELSE
                next_state<='0';
            END IF;
        ELSE
              next_state<=current_state;
        END IF;
    END IF;
END PROCESS NEXT_GET;
END Behavioral;
```

2. RUN_NET

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



ENTITY RUN_NET IS
GENERIC(r:INTEGER:=8;c:INTEGER:=8);
PORT (clk,srstn,ce,load,pre_load:IN STD_LOGIC;
      left_side:IN STD_LOGIC_VECTOR(r-1 DOWNTO 0);
      right_side:OUT STD_LOGIC_VECTOR(r-1 DOWNTO 0);
      read_cell:OUT STD_LOGIC_VECTOR(r*c-1 DOWNTO 0)
       );
END RUN_NET;
ARCHITECTURE Behavioral of RUN_NET IS
COMPONENT SINGLE_CELL
PORT (clk,srstn,ce,load,NW,N,NE,E,SE,S,SW,W:IN STD_LOGIC;
       state:OUT STD_LOGIC );
END COMPONENT;
SIGNAL NW,N,NE,E,SE,S,SW,W,cell:STD_LOGIC_VECTOR(R*C-1 DOWNTO 0);
BEGIN

 WIN: FOR i IN 0 TO r-1 GENERATE
         WITH pre_load SELECT
         W(i*c)<=  left_side(i)  WHEN '1' ,
                   cell((i+1)*c-1) WHEN '0' ,
                   'X'WHEN OTHERS;
 END GENERATE WIN;
     ------------------------------
---------internal part-----------
    CONC1:FOR i IN 1 TO r-2 GENERATE
    BEGIN
        CONC2:FOR j IN 1 TO c-2 GENERATE
        BEGIN
            NW(i*c+j)<=cell((i-1)*c+j-1);
            N(i*c+j)<=cell((i-1)*c+j);
            NE(i*c+j)<=cell((i-1)*c+j+1);
            W(i*c+j)<=cell(i*c+j-1);
            E(i*c+j)<=cell(i*c+j+1);
            SW(i*c+j)<=cell((i+1)*c+j-1);
            S(i*c+j)<=cell((i+1)*c+j);
            SE(i*c+j)<=cell((i+1)*c+j+1);
        END GENERATE CONC2;
    END GENERATE CONC1;
-------------left edge-----------------------------
    LEFT:FOR i IN 1 TO r-2 GENERATE
    BEGIN
            NW(i*c)<=cell(i*c-1);
            N(i*c)<=cell((i-1)*c);
            NE(i*c)<=cell((i-1)*c+1);
      --      W(i*c)<=cell(i*c+c-1);
            E(i*c)<=cell(i*c+1);
            SW(i*c)<=cell((i+2)*c-1);
            S(i*c)<=cell((i+1)*c);
            SE(i*c)<=cell((i+1)*c+1);
    END GENERATE LEFT;
-----------------right edge
    RIGHT:FOR i IN 2 TO r-1 GENERATE
        NW(i*c-1)<=cell((i-1)*c-2);
        N(i*c-1)<=cell((i-1)*c-1);
        NE(i*c-1)<=cell((i-2)*c);
        W(i*c-1)<=cell(i*c-2);
        E(i*c-1)<=cell((i-1)*c);
        SW(i*c-1)<=cell((i+1)*c-2);
        S(i*c-1)<=cell((i+1)*c-1);
        SE(i*c-1)<=cell(i*c);
    END GENERATE RIGHT;
```

```
----------------up edge---------------------
    UP:FOR i IN 1 TO c-2 GENERATE
        NW(i)<=cell((r-1)*c+i-1);
        N(i)<=cell((r-1)*c+i);
        NE(i)<=cell((r-1)*c+i+1);
        W(i)<=cell(i-1);
        E(i)<=cell(i+1);
        SW(i)<=cell(c+i-1);
        S(i)<=cell(c+i);
        SE(i)<=cell(c+i+1);
    END GENERATE UP;
---------------down edge--------------------
    DOWN:FOR i IN 1 TO c-2 GENERATE
        NW((r-1)*c+i)<=cell((r-2)*c+i-1);
        N((r-1)*c+i)<=cell((r-2)*c+i);
        NE((r-1)*c+i)<=cell((r-2)*c+i+1);
        W((r-1)*c+i)<=cell((r-1)*c+i-1);
        E((r-1)*c+i)<=cell((r-1)*c+i+1);
        SW((r-1)*c+i)<=cell(i-1);
        S((r-1)*c+i)<=cell(i);
        SE((r-1)*c+i)<=cell(i+1);
    END GENERATE DOWN;
-----------------north west---------------
NWCORNER: BLOCK
BEGIN
    NW(0)<=cell(c*r-1);
    N(0)<=cell((r-1)*c);
    NE(0)<=cell((r-1)*c+1);
  --  W(0)<=cell(c-1);
    E(0)<=cell(1);
    SW(0)<=cell(c*2-1);
    S(0)<=cell(c);
    SE(0)<=cell(c+1);
END BLOCK NWCORNER;
------------------north east------------------
NECORNER: BLOCK
BEGIN
    NW(c-1)<=cell(r*c-2);
    N(c-1)<=cell(r*c-1);
    NE(c-1)<=cell((r-1)*c);
    W(c-1)<=cell(c-2);
    E(c-1)<=cell(0);
    SW(c-1)<=cell(2*c-2);
    S(c-1)<=cell(2*c-1);
    SE(c-1)<=cell(c);
END BLOCK NECORNER;
-------------------------south west---------------------------
SWCORNER: BLOCK
BEGIN
    NW(c*(r-1))<=cell((c-1)*r-1);
    N(c*(r-1))<=cell((c-2)*r);
    NE(c*(r-1))<=cell((c-2)*r+1);
   --W(c*(r-1))<=cell(c*r-1);
    E(c*(r-1))<=cell((c-1)*r+1);
    SW(c*(r-1))<=cell(c-1);
    S(c*(r-1))<=cell(0);
    SE(c*(r-1))<=cell(1);
END BLOCK SWCORNER;
--------------------------------south east------------------------------
SECORNER: BLOCK
BEGIN
    NW(c*r-1)<=cell((r-1)*c-2);
    N(c*r-1)<=cell((r-1)*c-1);
    NE(c*r-1)<=cell((r-2)*c);
    W(c*r-1)<=cell(r*c-2);
    E(c*r-1)<=cell((r-1)*c);
    SW(c*r-1)<=cell(c-2);
    S(c*r-1)<=cell(c-1);
    SE(c*r-1)<=cell(0);
```

```
END BLOCK SECORNER;
-----------------------------------------
GENERATE_CELL1:FOR i IN 0 TO R-1 GENERATE
BEGIN
    GENERATE_CELL2:FOR j IN 0 TO C-1 GENERATE
        CELL_INTIA:SINGLE_CELL PORT MAP(clk=>clk,srstn=>srstn,ce=>ce,load=>load,NW=>NW(i*C+j),N=>N(i*C+j),
            NE=>NE(i*C+j),W=>W(i*C+j),E=>E(i*C+j),SW=>SW(i*C+j),S=>S(i*C+j),SE=>SE(i*C+j),state=>cell(i*C+j));
    END GENERATE GENERATE_CELL2;
END GENERATE GENERATE_CELL1;
-----------------------------------
OUTPUT:FOR i IN 0 TO r-1 generate
    right_side(i)<=cell(i*c+c-1);
END GENERATE OUTPUT;
read_cell<=cell;
END Behavioral;
```

3. RUN_SIM.vdh

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

ENTITY RUN_SIM IS
GENERIC(r:INTEGER:=8;c:INTEGER:=8);
END RUN_SIM;
ARCHITECTURE Behavioral OF RUN_SIM IS
COMPONENT RUN_NET
GENERIC(r:INTEGER:=8;c:INTEGER:=8);
PORT (clk,srstn,ce,load,pre_load:IN STD_LOGIC;
       right_side:OUT STD_LOGIC_VECTOR(r-1 DOWNTO 0);
       left_side:IN STD_LOGIC_VECTOR(r-1 DOWNTO 0);
       read_cell :OUT  STD_LOGIC_VECTOR(r*c-1 DOWNTO 0)
         );
END COMPONENT;
CONSTANT HPCLK:TIME:=10ns;
SIGNAL clk:STD_LOGIC:='0';
SIGNAL srstn:STD_LOGIC:='1';
SIGNAL ce:STD_LOGIC:='1';
SIGNAL load:STD_LOGIC:='0';
SIGNAL pre_load:STD_LOGIC:='0';
SIGNAL left_side:STD_LOGIC_VECTOR(r-1 DOWNTO 0);
SIGNAL right_side:STD_LOGIC_VECTOR(r-1 DOWNTO 0);
SIGNAL read_cell :STD_LOGIC_VECTOR(c*r-1 DOWNTO 0);
BEGIN
GENERATE_CLK:PROCESS
begin
    clk<='0';
    wait for HPCLK;
    clk<='1';
    wait for HPCLK;
END PROCESS GENERATE_CLK;
SIM:PROCESS
BEGIN
    WAIT FOR 2*HPCLK;
    srstn<='1';
    WAIT FOR 2*HPCLK;
    srstn<='0';
    WAIT FOR 2*HPCLK;
    srstn<='1';
    ce<=1;


    left_side<="11011111";
    pre_load<='1';
    load<='1';
    WAIT FOR 2*HPCLK;
    load<='0';
```

```
    pre_load<='0';
    WAIT FOR 100*HPCLK;

    left_side<="11111111";
    pre_load<='1';
    load<='1';
    WAIT FOR 2*HPCLK;
    load<='0';
    pre_load<='0';
    WAIT FOR 100*HPCLK;

    left_side<="11100111";
    pre_load<='1';
    load<='1';
    WAIT FOR 2*HPCLK;
    load<='0';
    pre_load<='0';
    WAIT FOR 10*HPCLK;


END PROCESS SIM;
RUN_NET_INTIA:RUN_NET PORT MAP(clk=>clk,srstn=>srstn,ce=>ce,load=>load,pre_load=>pre_load,
      right_side=>right_side,left_side=>left_side,read_cell=>read_cell);
END Behavioral;
```

4. SINGLE_CELL_SIM.vhd

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

ENTITY SINGLE_CELL_SIM IS
END SINGLE_CELL_SIM;
ARCHITECTURE Behavioral OF SINGLE_CELL_SIM IS
COMPONENT SINGLE_CELL
PORT (clk,srstn,ce,load,NW,N,NE,W,E,SW,S,SE:IN STD_LOGIC;
       state:OUT STD_LOGIC);
END COMPONENT;
CONSTANT HPCLK:TIME:=10ns;
SIGNAL clk:STD_LOGIC;
SIGNAL srstn:STD_LOGIC;
SIGNAL ce:STD_LOGIC;
SIGNAL load:STD_LOGIC;
SIGNAL NW,N,NE,W,E,SW,S,SE:STD_LOGIC;
SIGNAL state:STD_LOGIC;
GENERATE_CLK:PROCESS
BEGIN
    clk<='0';
    WAIT FOR HPCLK;
    clk<='1';
   WAIT FOR HPCLK;
END PROCESS GENERATE_CLK ;
BEGIN
SIM_CELL:PROCESS
BEGIN
    ce<='1';
    srstn<='0';
    WAIT FOR 2*HPCLK;
    srstn<='1';
    WAIT FOR 2*HPCLK;
    srstn<='0';

    WAIT FOR 2*HPCLK;
    load<='1';
    WAIT FOR 2*HPCLK;
    load<='0';

    WAIT FOR 6*HPCLK;
    NW<='1';
```

```
    N<='0';
    NE<='0';
    W<='1';
    E<='1';
    SW<='0';
    S<='0';
    SE<='0';
    WAIT FOR 6*HPCLK;
END PROCESS SIM_CELL;
CELL_INTIA:SINGLE_CELL PORT MAP(clk=>clk,srstn=>srstn,ce=>ce,load=>load,NW=>NW,
                          N=>N,NE=>NE,W=>W,E=>E,SW=>SW,S=>S,SE=>SE,state=>state);
END Behavioral;
```