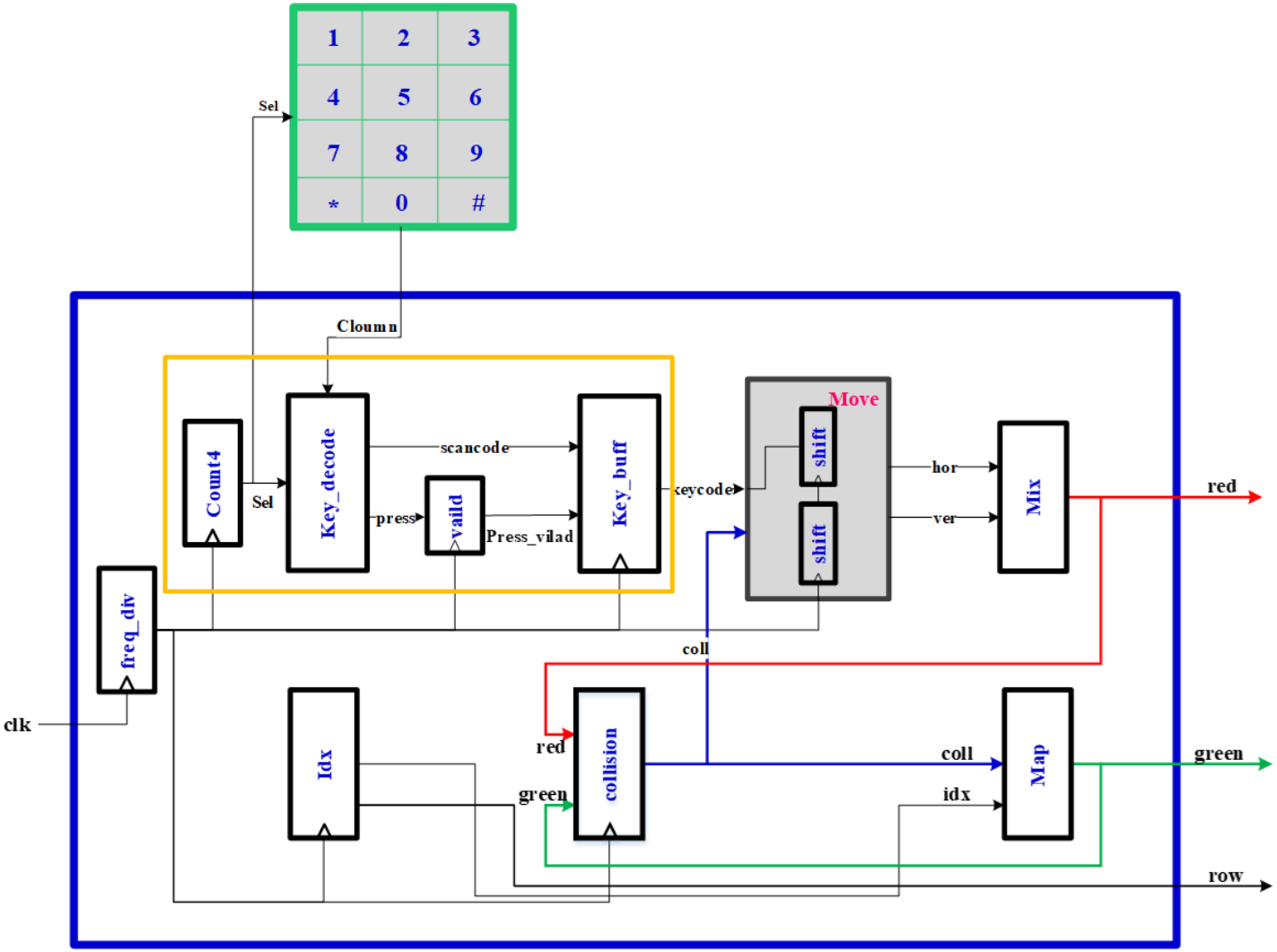
FPGA Lab-08

Maze

Lab content

- Using the keyboard as input, control the little red dots displayed on the 8x8 dot matrix to complete the maze without collisions.

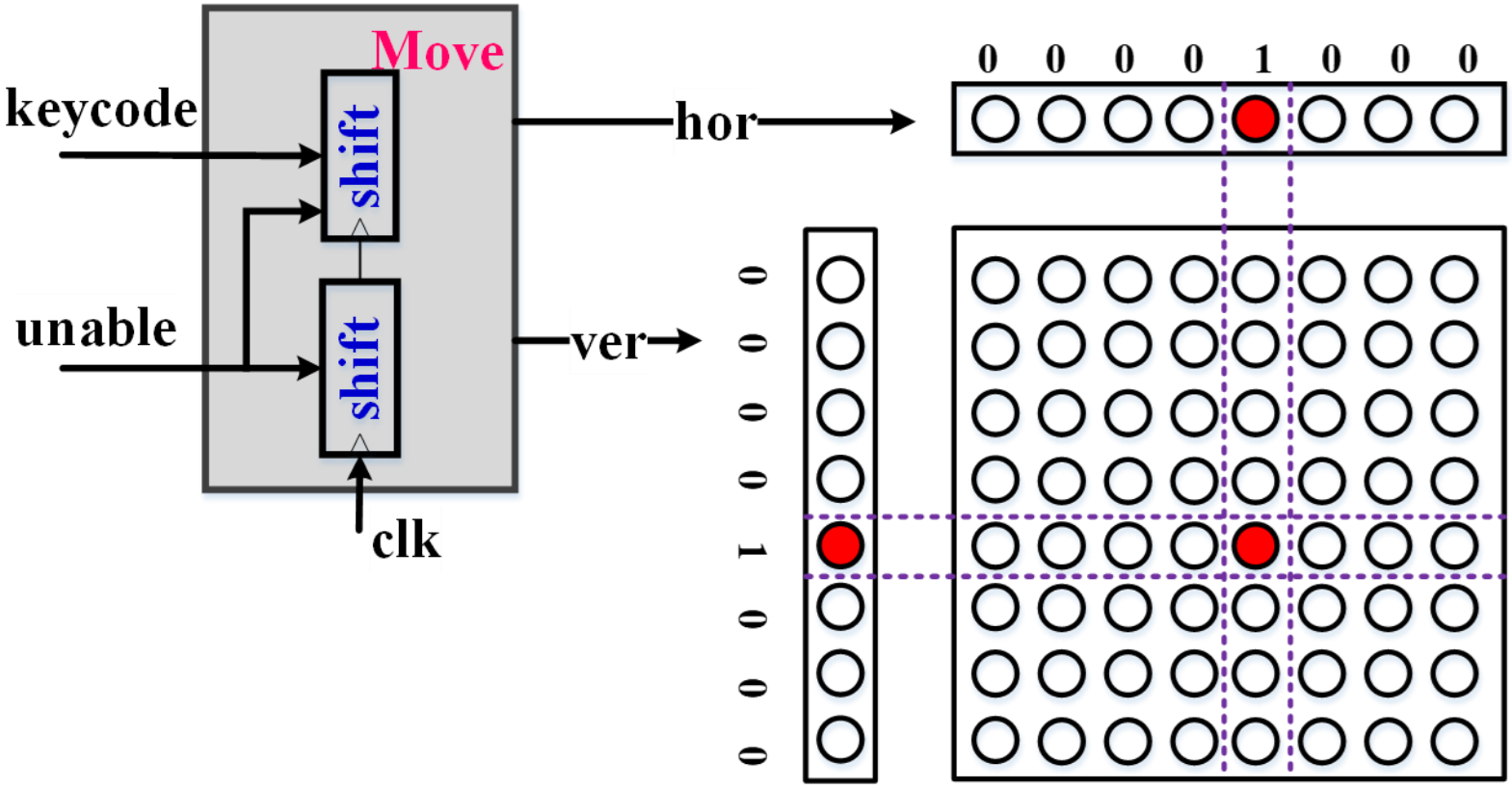
System Block Diagram



Verilog Codes

- collision.v
- map.v
- move.v
- mix.v
- idx.v (same as previous idx_cnt and row_gen is slightly different from previous)
- shift.v
- vaild.v (please write your code) (lab6_debounce_ctl.v)
- key_decode.v(please write your own code, slightly different from lab 6)
- key_buf.v(please write your own code, slightly different from lab 6)
- count6.v (please write your own code)
- freq_div.v(please write your own code)
- red_dot_top.v

MOVE(1/2)

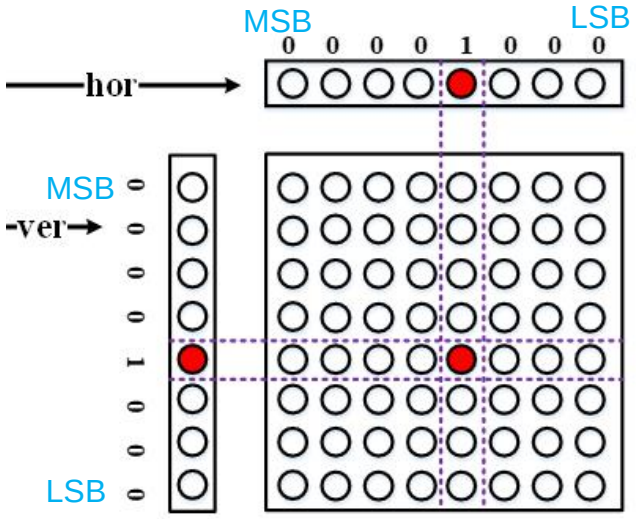

Move
keycode
unable
shift
shift
clk
hor
ver
0 0 0 0 1 0 0 0
0
0
0
0
1
0
0
0

MOVE(2/2)

0000_1000	0000_1000
ver	hor

Original Position	Go up	Original Position	Go down
0	0	0	0
0	0	0	0
0	0	0	0
0	1	0	0
1	0	1	0
0	0	0	1
0	0	0	0
0	0	0	0

Original Position	0	0	0	0	1	0	0	0
Go Right	0	0	0	0	0	1	0	0
Original Position	0	0	0	0	1	0	0	0
Go Left	0	0	0	1	0	0	0	0



VerilogCode -shift.v

```
module      shift(left, right, reset, unable, out, clk);
input       left, right, reset, clk, unable;
output reg  [7:0]out;

always@(posedge clk or posedge reset)
begin
    if(reset)
        out<=8'b0000_1000;
    else if(unable)                 //collision happen
        out<=8'b0000_0000;
    else if(left)
        ~~~your code~~~
    else if(right)
        ~~~your code~~~
    else
        out<=out;
end
endmodule
```

VerilogCode -move.v

- module move(reset, unable, keycode, ver, hor, clk);
- input reset, clk, unable;
- input [3:0]keycode;
- output [7:0]ver, hor;
- wire left, right, up, down;

- assign left =~keycode[1]& keycode[2];
- assign right = keycode[1]& keycode[2];
- assign up = keycode[1]&~keycode[2];
- assign down= keycode[3];

- shift S1(~~~your code~~~); //left & right
- shift S2(~~~your code~~~); //up & down

- endmodule

The keycode is the value buffered by the scancode, i.e. the number scanned, i.e. 2,4,6,8.

2 ,0010---up
4 ,0100---left
6 ,0110---right
8 ,1000---down

VerilogCode -map.v

```
module          map(addr,data);
input           [3:0]addr;
output reg      [7:0]data;
always@(addr)
begin
case(addr)
        4'd0    :data=      //Create your own map
        4'd1    :data=
        4'd2    :data=
        4'd3    :data=
        4'd4    :data=
        4'd5    :data=
        4'd6    :data=
        4'd7    :data=

        4'd8    :data=8'b1111_1111;
        4'd9    :data=8'b1111_1111;
        4'd10   :data=8'b1111_1111;
        4'd11   :data=8'b1111_1111;
        4'd12   :data=8'b1111_1111;
        4'd13   :data=8'b1111_1111;
        4'd14   :data=8'b1111_1111;
        4'd15   :data=8'b1111_1111;
        default :data=8'b0000_0000;
endcase
end
endmodule
```

Control signal

```
addr = { coll, idx };
```

addr={coll=0, idx}

0	000
0	001
0	010
0	011
0	100
0	101
0	110
0	111

Map 1

addr={coll=1, idx}

1	000
1	001
1	010
1	011
1	100
1	101
1	110
1	111

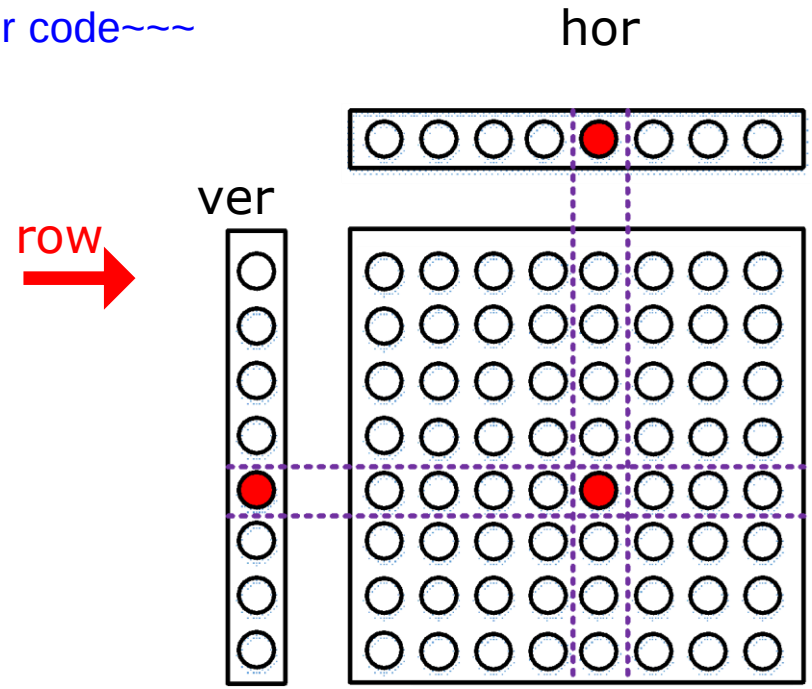
Map 2

VerilogCode -idx.v

```
module          idx(clk, reset, idx, row);
input            reset, clk;
output reg      [2:0]idx;
output reg      [7:0]row;
always@(posedge clk or posedge reset)
begin
        if(reset) begin
                idx<=3'b000;
                row<=8'b1000_0000;
        end
        else begin
                idx<=idx+3'b001;
                row<={row[0],row[7:1]};
        end
end
endmodule
```

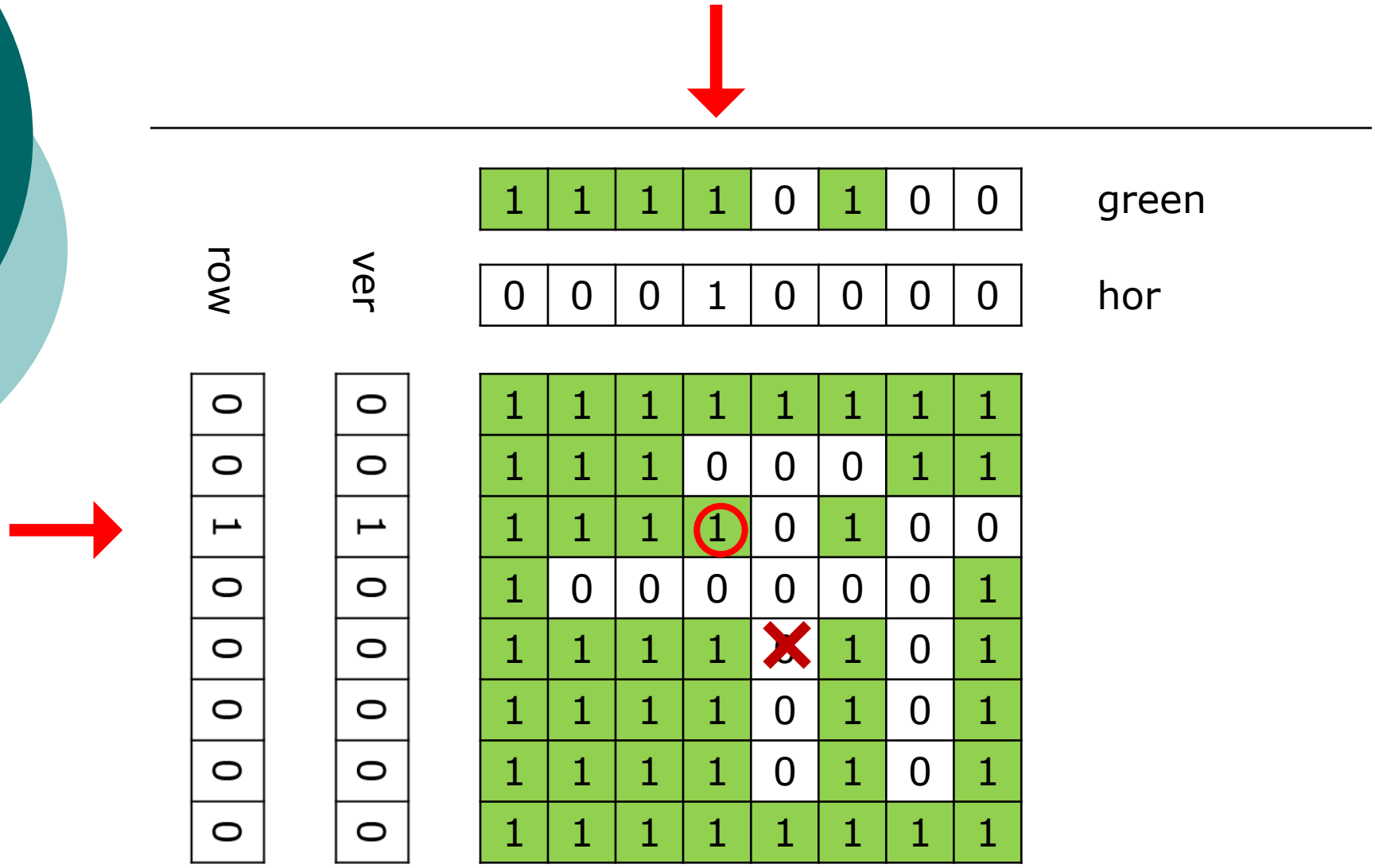
VerilogCode -mix.v

- module mix(ver, hor, row, red);
- input [7:0]ver, hor, row;
- output [7:0]red;
- assign red=~~~your code~~~
- endmodule



VerilogCode -collision.v

```
module          collision(clk, reset, red, green, coll);
input           clk, reset;
input           [7:0]red, green;
output reg      coll;
always@(posedge clk or posedge reset)
begin
        if(reset)
                coll<=1'b0;
        else if((red & green) != 8'b0)    //Collision happen
                coll<=1'b1;
        else
                coll<=coll;
end
endmodule
```


1 1 1 1 0 1 0 0
green
0 0 0 1 0 0 0 0
hor
row
ver
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
1 1 1 1 1 1 1 1
1 1 1 0 0 0 1 1
1 1 1 1 0 1 0 0
1 0 0 0 0 0 0 1
1 1 1 1 0 1 0 1
1 1 1 1 0 1 0 1
1 1 1 1 0 1 0 1
1 1 1 1 1 1 1 1

row	{coll,cnt}	ver	hor	green	red	coll
1000_0000	{0,000}	00100000	00010000	11111111	00000000	0
0100_0000	{0,001}	00100000	00010000	11100011	00000000	0
0010_0000	{0,010}	00100000	00010000	11110100	00010000	1
0001_0000	{0,011}	00100000	00010000	10000001	00000000	0
0000_1000	{0,100}	00100000	00010000	11110101	00000000	0
0000_0100	{0,101}	00100000	00010000	11110101	00000000	0
0000_0010	{0,110}	00100000	00010000	11110101	00000000	0
0000_0001	{0,111}	00100000	00010000	11111111	00000000	0

1`	1	1	1	1	1	1	1
1	1	1	0	0	0	1	1
1	1	1	1	0	1	0	0
1	0	0	0	0	0	0	1
1	1	1	1	0	1	0	1
1	1	1	1	0	1	0	1
1	1	1	1	0	1	0	1
1	1	1	1	1	1	1	1

Red_dot_top.v

- module red_dot_top(clk, row, red, green, column, sel, reset);

- input reset, clk;
- input [2:0]column; //AA13,AB12,Y16
- output [7:0]red, row, green;
  //D7,D6,A9,C9,A8,C8,C11,B11
  //T22,R21,C6,B6,B5,A5,B7,A7
  //A10,B10,A13,A12,B12,D12,A15,A14
- output [2:0]sel; AB10,AB11,AA12
- wire ck, press, press_vaild, coll;
- wire [3:0]keycode, scancode, addr;
- wire [2:0]idx;
- wire [7:0]hor, ver;

- assign addr = { coll, idx };

- key_decode M1 (~~~your code~~~);
- key_buff M2 (~~~your code~~~);
- vaild M3 (~~~your code~~~);
- count6 M4 (~~~your code~~~);
- move M5 (~~~your code~~~);
-
- freq_div#(14) M6 (~~~your code~~~);

- map M7 (~~~your code~~~);
- idx M8 (~~~your code~~~);
- mix M9 (~~~your code~~~);
- collision M10 (~~~your code~~~);

- endmodule