Computer Architecture

computer assignment 1

8_Queen_algorithm_RTL Design

Name: Mohammad Moghiseh

Date: 29 October 2024

In this computer exercise, you are asked to find a hardware to implement the 8_Queen algorithm in chess.

You have to place 8 Queens on the chessboard in such a way that no two one threaten each other.

The Python code below shows an implementation of this algorithm, which is implemented using the backtracking method:

-the algorithm starts by placing a Queen in the first column, then moves to the next column and places a Queen in the first safe row of that column.

- If the algorithm reaches the eighth column and all the Queen are in their safe positions, the algorithm ends.

- If the algorithm cannot place a queen in a safe position in a particular column, it will return to the previous column and try another row to place that queen.

- The (is_safe) function checks whether it is safe to place a Queen on a specific row and column that is given in the input.

```python
def solve_n_queens (n ,board , col ) :
    if col == n :
        for o in range(n) :
            print(board[o])
        return True
    for i in range (n) :
        if is_safe(n , board , i , col) :
            board [i][col] = 1
            if solve_n_queens(n , board , col +1) :
                return True
            board [i][col] = 0
    return False
```

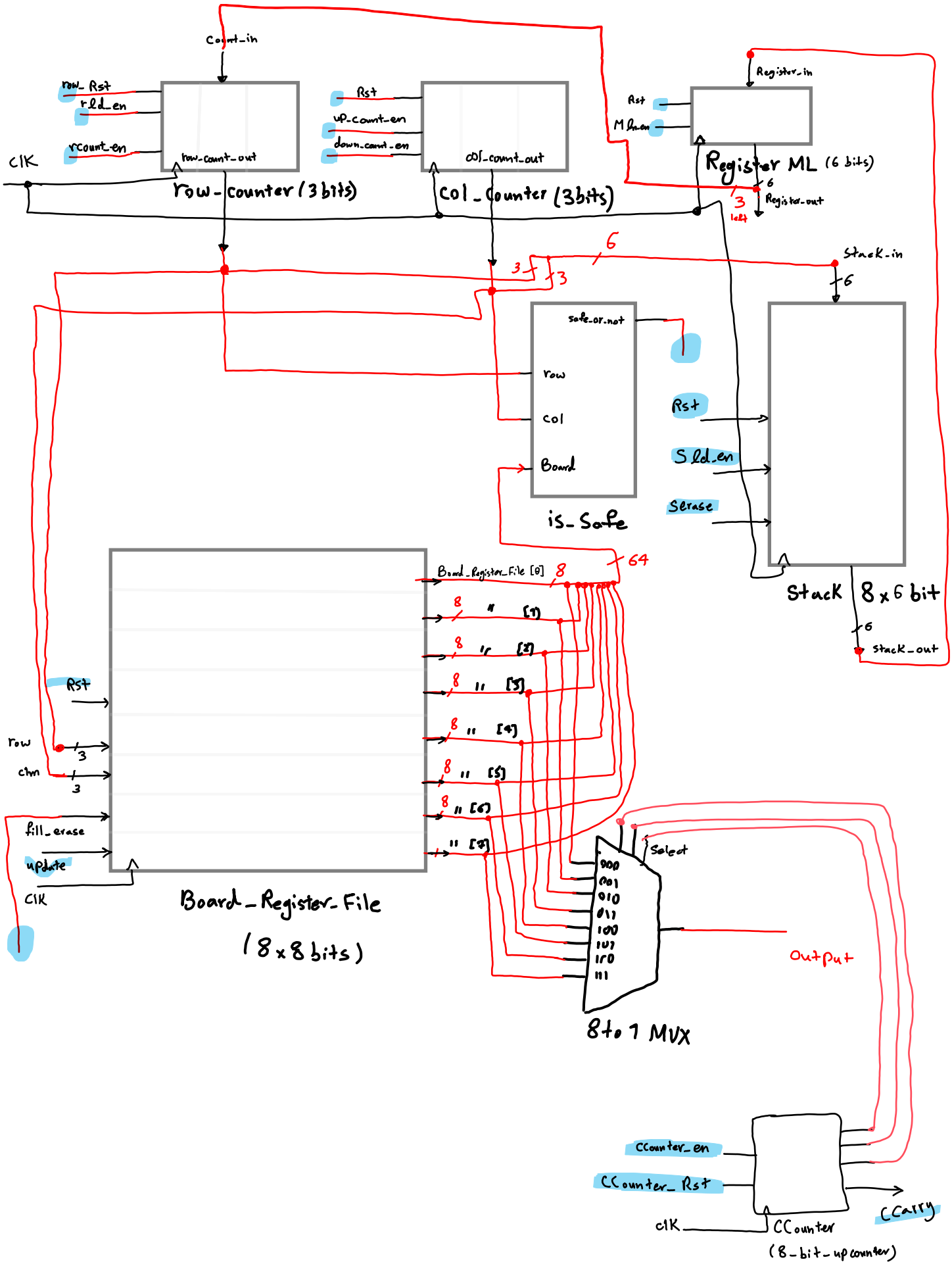
```python
def is_safe(n, board , row , col) :
    q=row
    l=col
    for m in range(n) :
        if board[row][m] ==1 :
            return False
    for p in range(n) :
        if board[p][col] ==1 :
            return False
    for i in range(4) :
        q=row
        l=col
        while (0 <= q < n) and (0 <= l < n) :
            if board[q][l] == 1 :
                return False
            else :
                if i==0 :
                    q = q+ 1
                    l = l+1
                elif i==1:
                    q = q + 1
                    l = l-1
                elif i==2 :
                    q = q- 1
                    l = l+1
                else :
                    q = q- 1
                    l = l-1
    return True

n=8
board = [([0]*n) for i in range(n)]
solve_n_queens (n ,board , 0 )
```

result in python:

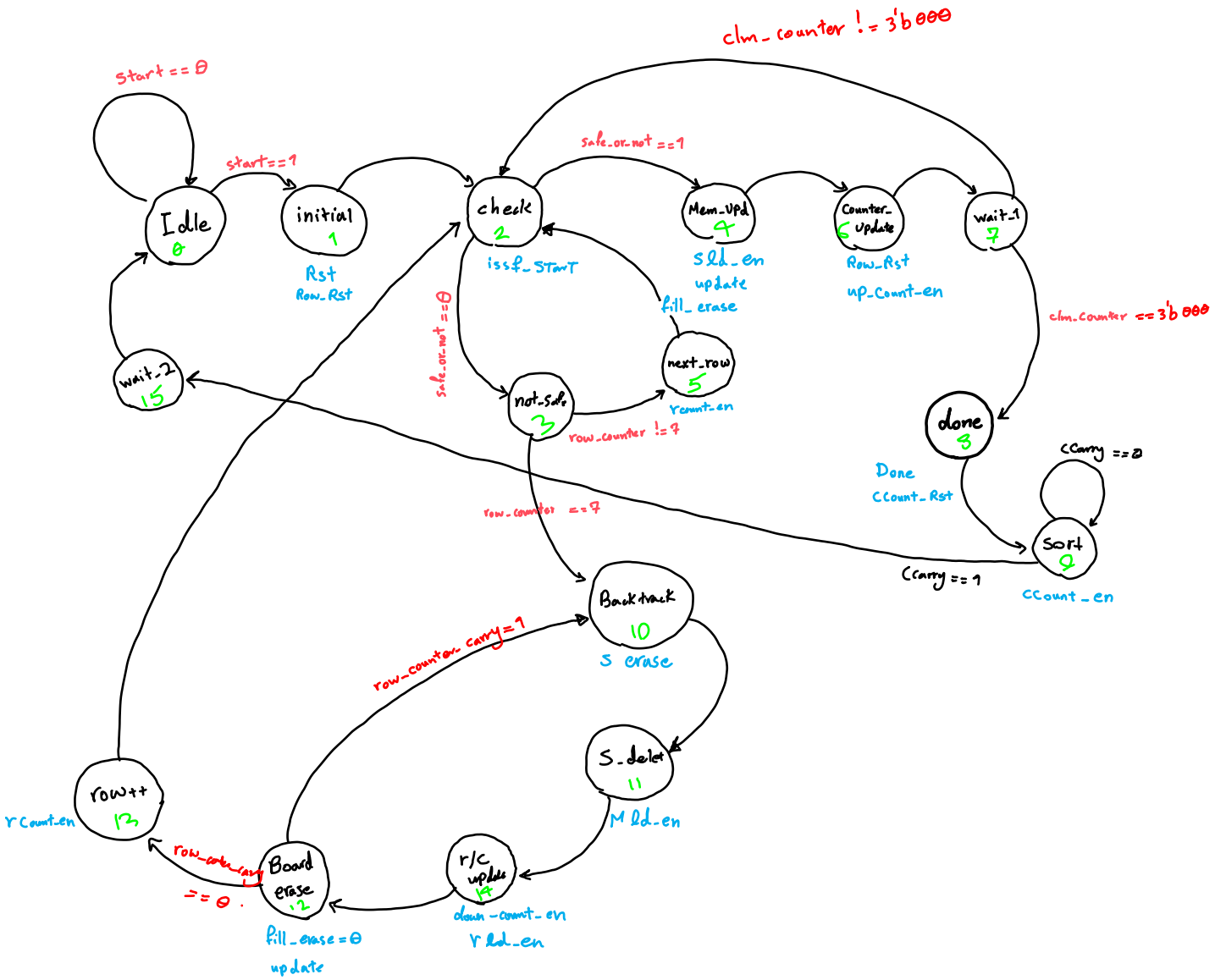
```
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
```

now we go for design it as a hardware.

DATA PATH : what



CONTROLLER : when



state assignment

* درباره شمارنده CCounter دقت کنید وقتی مقدار 7 را داشته باشد، سیگنال CCarry را نیز ایشو می کند!

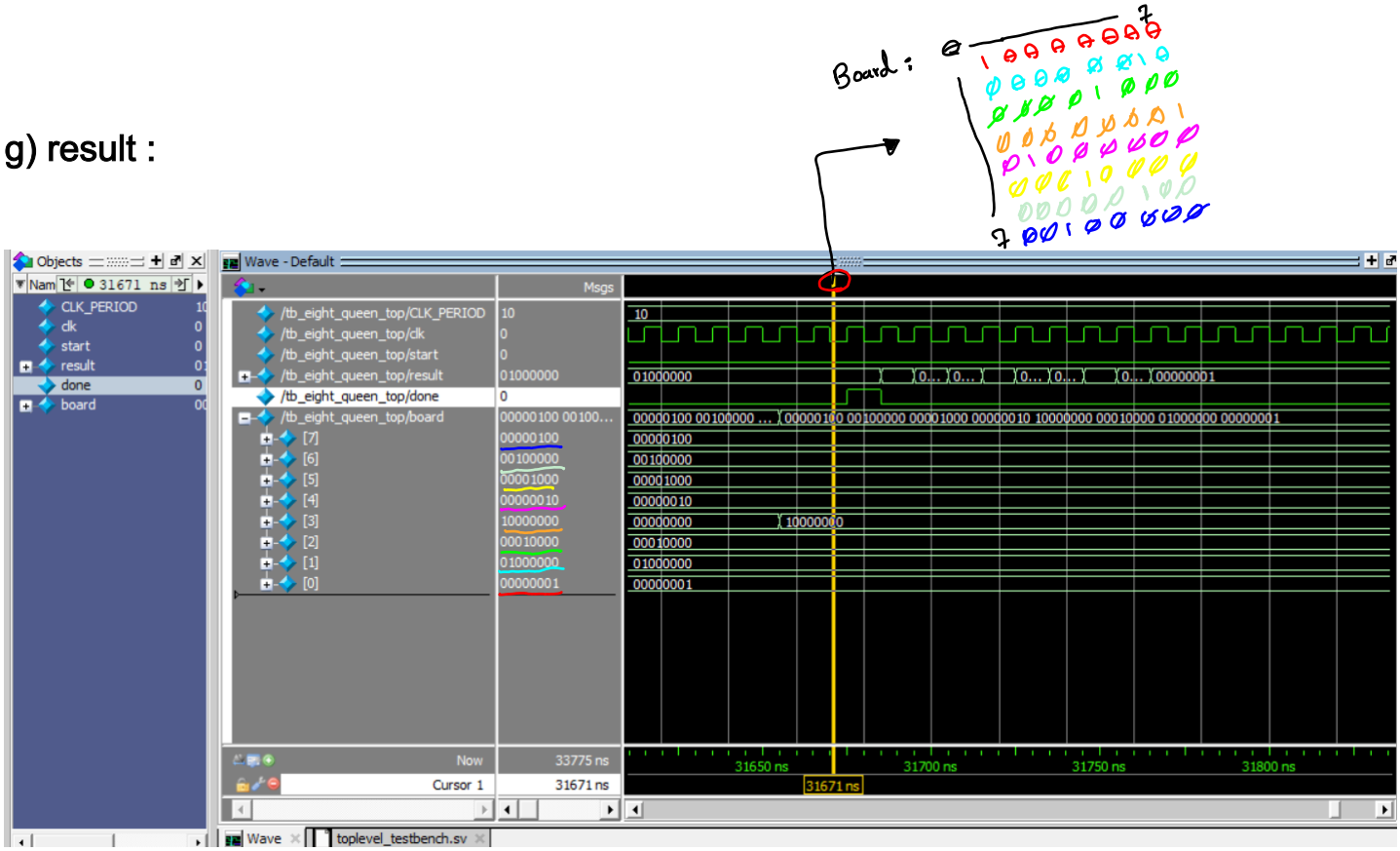
Verilog description for top-level 8-queen module:

```
Ln#
 1  `timescale 1ns/1ns
 2
 3  module eight_queen_top (
 4      input logic clk,
 5      input logic start,
 6      output logic [7:0] result,
 7      output logic done
 8  );
 9
10      // Wires for interconnecting the FSM and datapath
11      logic rst;
12      logic [2:0]clm_counter;
13      logic rld_en;
14      logic row_rst;
15      logic update;
16      logic fill_erase;
17      logic up_count_en;
18      logic down_count_en;
19      logic sld_en;
20      logic serase;
21      logic ccount_rst;
22      logic ccount_en;
23      logic rcount_en;
24      logic mld_en;
25      logic safe_or_not;
26      logic clm_counter_carry;
27      logic row_counter_carry;
28      logic ccarry;
29      logic [7:0]board[7:0];
30
31      // Instantiate the FSM controller
32      eight_queen_fsm controller (
33          .clk(clk),
34          .start(start),
35          .safe_or_not(safe_or_not),
36          .clm_counter_carry(clm_counter_carry),
37          .row_counter_carry(row_counter_carry),
38          .ccarry(ccarry),
39          .rst(rst),
40          .rld_en(rld_en),
41          .row_rst(row_rst),
42          .update(update),
43          .fill_erase(fill_erase),
44          .up_count_en(up_count_en),
45          .down_count_en(down_count_en),
46          .sld_en(sld_en),
47          .serase(serase),
48          .ccount_rst(ccount_rst),
49          .ccount_en(ccount_en),
50          .done(done),
51          .rcount_en(rcount_en),
52          .mld_en(mld_en),
53          .clm_counter(clm_counter)
54      );
55
56      // Instantiate the datapath
57      eigth_queen_datapath datapath (
58          .clk(clk),
59          .row_rst(row_rst),
60          .rld_en(rld_en),
61          .rcount_en(rcount_en),
62          .rst(rst),
63          .update(update),
64          .fill_erase(fill_erase),
65          .up_count_en(up_count_en),
66          .down_count_en(down_count_en),
67          .mld_en(mld_en),
68          .sld_en(sld_en),
69          .serase(serase),
70          .ccounter_rst(ccount_rst),
71          .ccounter_en(ccount_en),
72          .result(result),
73          .ccarry(ccarry),
74          .safe_or_not(safe_or_not),
75          .row_carry(row_counter_carry),
76          .clm_carry(clm_counter_carry),
77          .board(board),
78          .clm_counter(clm_counter)
79      );
80
81  endmodule
82
83
```

Verilog test bench for top-level 8-queen module:

```
Ln#
 1  `timescale 1ns/1ns
 2
 3  module tb_eight_queen_top;
 4
 5      // Parameters
 6      parameter CLK_PERIOD = 10; // Clock period in ns
 7
 8      // Testbench signals
 9      logic clk;
10      logic start;
11      logic [7:0] result;
12      logic done;
13
14      // Instantiate the top-level module
15      eight_queen_top uut(
16          .clk(clk),
17          .start(start),
18          .result(result),
19          .done(done)
20      );
21
22      // Clock generation
23      initial begin
24          clk = 0;
25          forever #(CLK_PERIOD / 2) clk = ~clk; // Toggle clock every half period
26      end
27
28      // Test sequence
29      initial begin
30          // Initialize inputs
31          start = 0;
32          // Wait for a few clock cycles
33          #(CLK_PERIOD * 5);
34          // Start the algorithm
35          start = 1;
36          #(CLK_PERIOD); // Wait one clock cycle
37          // Release start signal
38          start = 0;
39          // Wait for the algorithm to finish
40          wait(done); // Wait until done signal is asserted
41          // Display the result
42          $display("Result: %b", result);
43          // End the simulation after observing the output
44          #(CLK_PERIOD * 10);
45          $finish;
46      end
47  endmodule
```

g) result :

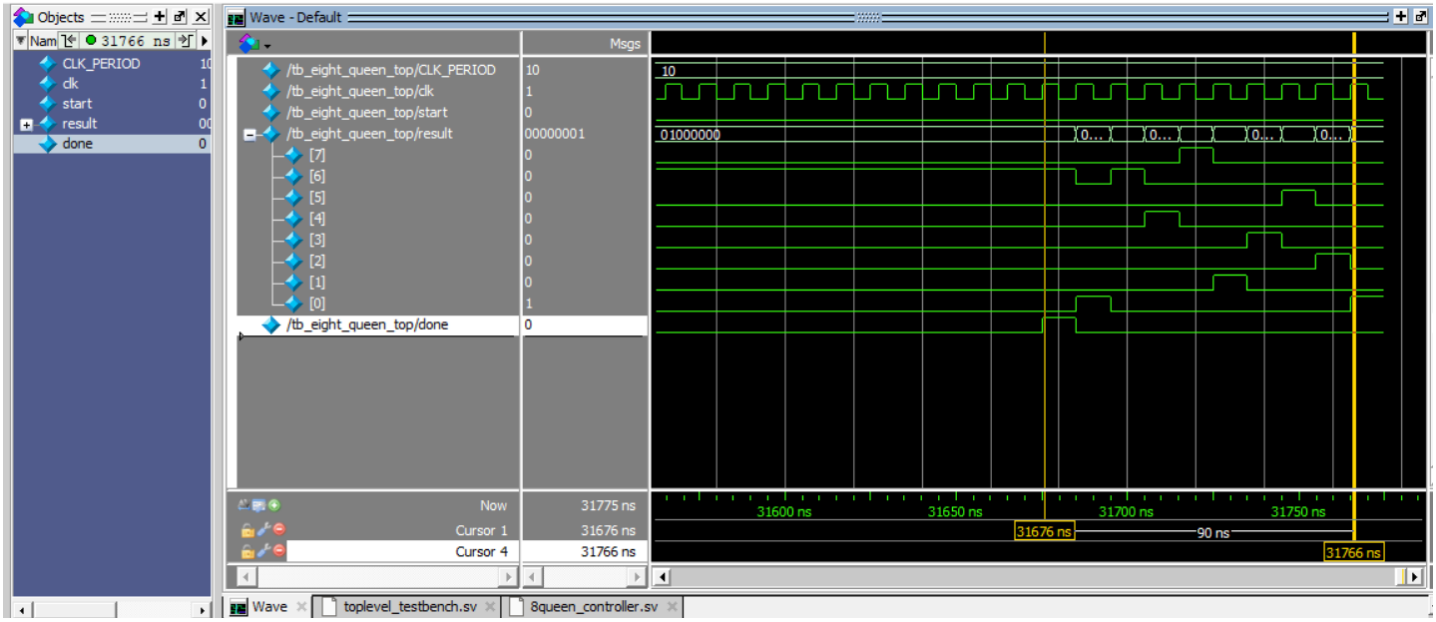


we got done signal after 3168 clock period!

and we adjust the clock period 10ns...

we got the exact value which were found out in part one using python code but in 8 clock and each row of board :

in the CA description they said that we dont need whole of board regidter but here we also pass board as output of our top module to better realize the accuracy of result.
if you dont want it so delete it from output of module as bellow:



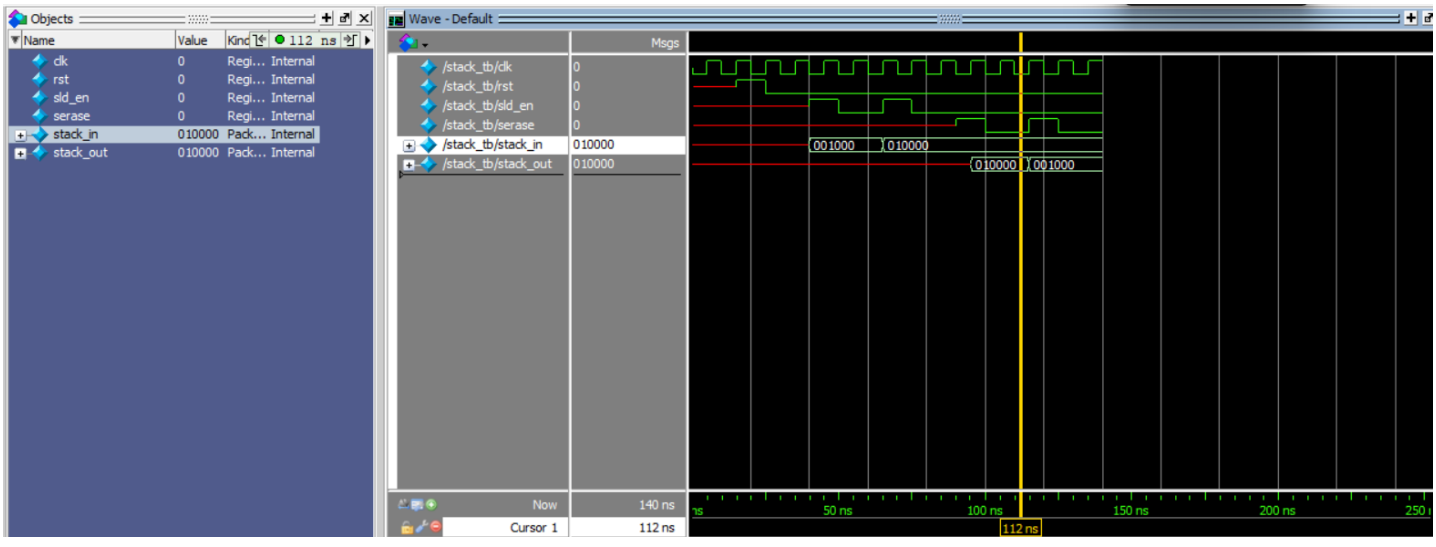
thank you for your attention

you can see the detail of controller and datapath as below :

datapath design: stack 8*6bits

```
Ln#                                                                    112 ns
 1  `timescale 1ns/1ns
 2  module stack8_6(input clk,rst,sld_en,serase,input [5:0]stack_in,output logic [5:0]stack_out);
 3  logic [2:0] index;
 4  logic [5:0] temp[0:7];
 5  logic stack_full,stack_empty;
 6  always @ (posedge clk)begin
 7          if (rst==1)begin
 8                  stack_empty =1;
 9                  index=3'b000;
10                  stack_full=0;
11          end
12          if(sld_en == 1 && stack_full==0)begin
13                  stack_empty <=0;
14                  index=index+1;
15                  temp[index] <= stack_in;
16                  if (index==3'b111) stack_full<=1;
17          end
18          else if(serase == 1 && 1)begin
19                  stack_full <=0;
20                  stack_out <= temp[index];
21                  index=index-1;
22                  if (index==3'b000) stack_empty<=1;
23          end
24  end
25  endmodule
```

its testbench:



datapath design: safe or not combinational logic

```
Ln#
1  module safe_or_not_dd(input [2:0]row_counter,clm_counter  ,input [7:0]board[7:0], output logic safe_or_not);
2          integer i, j;
3          integer m,n;
4          logic conflict;
5  //is_safe_module_design:
6          always @(*) begin
7                  conflict = 0;
8                  // Check the entire row for another queen
9                  for (m = 0; m < 8; m = m + 1) begin
10                         if (board[row_counter][m] == 1 && m != clm_counter) begin
11                                 conflict = 1;
12                         end
13                 end
14                 // Check the entire column for another queen
15                 for (n = 0; n < 8; n = n + 1) begin
16                         if (board[n][clm_counter] == 1 && n != row_counter) begin
17                                 conflict = 1;
18                         end
19                 end
20                 // Check all four diagonal directions
21                 // (1) Down-right diagonal
22                 m = row_counter; n = clm_counter;
23                 while (m < 8 && n < 8) begin
24                         if (board[m][n] == 1 && (m != row_counter || n != clm_counter)) conflict = 1;
25                         m = m + 1;
26                         n = n + 1;
27                 end
28                 // (2) Down-left diagonal
29                 m = row_counter; n = clm_counter;
30                 while (m < 8 && n >= 0) begin
31                         if (board[m][n] == 1 && (m != row_counter || n != clm_counter)) conflict = 1;
32                         m = m + 1;
33                         n = n - 1;
34                 end
35                 // (3) Up-right diagonal
36                 m = row_counter; n = clm_counter;
37                 while (m >= 0 && n < 8) begin
38                         if (board[m][n] == 1 && (m != row_counter || n != clm_counter)) conflict = 1;
39                         m = m - 1;
40                         n = n + 1;
41                 end
42                 // (4) Up-left diagonal
43                 m = row_counter; n = clm_counter;
44                 while (m >= 0 && n >= 0) begin
                    if (board[m][n] == 1 && (m != row_counter || n != clm_counter)) conflict = 1;
                    m = m - 1;
                    n = n - 1;
                end
                safe_or_not = ~conflict;
        end
endmodule
```

testbench : I have checked it

datapath design: board 8*8 register

C:/Users/USER/Desktop/Fall2024/Computer Architecture/CA/CA1(8_queen_problem_accelerator)/board_datapath.sv - Default

```
Ln#
 1  `timescale 1ns/1ns
 2  module board_8_8(input [2:0]row_counter,input [2:0]clm_counter,input clk,rst,update,fill_erase,output logic [7:0]board[7:0] );
 3          integer i,j;
 4          logic [7:0]temp_board[7:0];
 5          assign board=temp_board;
 6          always @(posedge clk) begin
 7                  if (rst) begin
 8                          for (i = 0; i < 8; i = i + 1) begin
 9                                  for (j = 0; j < 8; j = j + 1) begin
10                                          temp_board[i][j] <= 1'b0; // Clear all cells
11                                  end
12                          end
13                  end
14                  else if (update==1 && fill_erase==1 ) begin
15                          temp_board[row_counter][clm_counter] <= 1'b1;
16                  end
17                  else if(update==1 && fill_erase==0 ) begin
18                          temp_board[row_counter][clm_counter] <= 1'b0;
19                  end
20          end
21  endmodule
```

datapath design: instansiation from datapath module and design counters and mux

```
Ln#
 1  `timescale 1ns/1ns
 2  module eigth_queen_datapath(input clk,input row_rst,rld_en,rcount_en,rst,update,fill_erase,up_count_en,down_count_en,mld_en,sld_en,serase,ccounter_rst,ccounter_en,output logic[7:0]result,
 3          output logic ccarry,safe_or_not,row_carry,clm_carry,output logic [7:0]board[7:0],output logic [2:0]clm_counter);
 4          logic [5:0]ml_reg_out;
 5          logic [2:0]row_counter;
 6          logic [2:0]ccounter;
 7          logic [5:0]stack_out;
 8          logic [5:0]stack_in;
 9          safe_or_not_dd safe_instanse(row_counter,clm_counter ,board,safe_or_not);
10          //row_counter_design:
11          always@(posedge clk)begin
12                  if (row_rst==1) row_counter<=3'b000;
13                  else if(rld_en==1) row_counter<=ml_reg_out[5:3];
14                  else if(rcount_en==1) row_counter<=row_counter+1;
15                  else row_counter<=row_counter;
16          end
17          assign row_carry=(row_counter==3'b111)?1:0;
18          //clm_counter_design:
19          always@(posedge clk)begin
20                  if(rst==1) clm_counter<=3'b000;
21                  else if(up_count_en==1) clm_counter<=clm_counter+1;
22                  else if(down_count_en==1) clm_counter<=clm_counter-1;
23                  else clm_counter<=clm_counter;
24          end
25          assign clm_carry=(clm_counter==3'b111)?1:0;
26          //ML_register_design:
27          always@(posedge clk)begin
28                  if(rst==1) ml_reg_out<=3'b000;
29                  else if(mld_en==1) ml_reg_out<=stack_out;
30                  else ml_reg_out<=ml_reg_out;
31          end
32          //stack instansiation and connecting:
33          assign stack_in={row_counter,clm_counter};
34          stack8_6 stackk(clk,rst,sld_en,serase,stack_in,stack_out);
35          //board instansiation:
36          board_8_8 boardd(row_counter,clm_counter,clk,rst,update,fill_erase,board);
37          //control_unit_counter_design:
38          always@(posedge clk)begin
39                  if(ccounter_rst==1) ccounter<=3'd0;
40                  else if(ccounter_en==1) ccounter<=ccounter+1;
41                  else ccounter<=ccounter;
42          end
43          assign ccarry=(ccounter==3'b111)?1:0;
44          //mux_for_put_on_output:
45                  case (ccounter)
46                          3'b000: result = board[0];
47                          3'b001: result = board[1];
48                          3'b010: result = board[2];
49                          3'b011: result = board[3];
50                          3'b100: result = board[4];
51                          3'b101: result = board[5];
52                          3'b110: result = board[6];
53                          3'b111: result = board[7];
54                          default: result = board[1]; // Default case
55                  endcase
56          end
57  endmodule
58
```

controller:

```
Ln#
  1  module eight_queen_fsm (
  2          input logic [2:0]clm_counter,
  3          input logic clk,
  4          input logic start,
  5          input logic safe_or_not,
  6          input logic clm_counter_carry,
  7          input logic row_counter_carry,
  8          input logic ccarry,
  9          output logic rst,
 10          output logic rld_en,
 11          output logic row_rst,
 12          output logic update,
 13          output logic fill_erase,
 14          output logic up_count_en,
 15          output logic down_count_en,
 16          output logic sld_en,
 17          output logic serase,
 18          output logic ccount_rst,
 19          output logic ccount_en,
 20          output logic done,
 21          output logic rcount_en,
 22          output logic mld_en
 23  );
 24
 25      // Define the states
 26          parameter [3:0] IDLE=4'd0, INITIAL=4'd1, CHECK=4'd2,NOT_SAFE=4'd3, MEM_UPD=4'd4, NEXT_ROW=4'd5,COUNTER_UPDATE=4'd6, WAIT_1=4'd7,
 27          DONE=4'd8,SORT=4'd9, BACKTRACK=4'd10, S_DELETE=4'd11,BOARD_ERASE=4'd12, ROW_INC=4'd13, RC_UPDATE=4'd14,WAIT_2=4'd15;
 28
 29      logic [3:0] state, next_state;
 30
 31      // State transition logic
 32      always @(posedge clk ) begin
 33              state <= next_state;
 34      end
 35
 36      // Next state and output logic
 37      always @(*) begin
 38          // Default values for outputs
 39          rld_en = 0;
 40          row_rst = 0;
 41          update = 0;
 42          fill_erase = 0;
 43          up_count_en = 0;
 44          down_count_en = 0;
 45            serase = 0;
 46            ccount_rst = 0;
 47            ccount_en = 0;
 48            done = 0;
 49            rst =0 ;
 50            rcount_en=0;
 51            mld_en=0;
 52
 53            case (state)
 54                IDLE: begin
 55                    if (start)
 56                        next_state = INITIAL;
 57                    else
 58                        next_state = IDLE;
 59                end
 60
 61                INITIAL: begin
 62                    row_rst = 1;
 63                    rst =1;
 64                    next_state = CHECK;
 65                end
 66
 67                CHECK: begin
 68                    if (safe_or_not)
 69                        next_state = MEM_UPD;
 70                    else
 71                        next_state = NOT_SAFE;
 72                end
 73
 74                NOT_SAFE: begin
 75                    if (row_counter_carry)
 76                        next_state = BACKTRACK;
 77                    else
 78                        next_state = NEXT_ROW;
 79                end
 80
 81                MEM_UPD: begin
 82                    sld_en = 1;
 83                    update = 1;
 84                    fill_erase = 1;
 85                    next_state = COUNTER_UPDATE;
 86                end
```

```
            NEXT_ROW: begin
                rcount_en = 1;
                next_state = CHECK;
            end

            COUNTER_UPDATE: begin
                row_rst=1;
                up_count_en=1;
                next_state = WAIT_1;
            end

            WAIT_1: begin
                if (clm_counter==3'b000)
                    next_state = DONE;
                else
                    next_state = CHECK;
            end

            DONE: begin
                done = 1;
                ccount_rst = 1;
                next_state = SORT;
            end

            SORT: begin
                ccount_en = 1;
                if (ccarry)
                    next_state = WAIT_2;
                else
                    next_state = SORT;
            end

            BACKTRACK: begin
                serase = 1;
                next_state = S_DELETE;
            end

            S_DELETE: begin
                mld_en = 1;
                next_state = RC_UPDATE;
            end

            BOARD_ERASE: begin
                    fill_erase = 0;
                    update = 1;
                    if (row_counter_carry==1)
                            next_state = BACKTRACK;
                    else
                            next_state = ROW_INC;

                end

                ROW_INC: begin
                    rcount_en = 1;
                    next_state = CHECK;
                end

                RC_UPDATE: begin
                    down_count_en = 1;
                    rld_en = 1;
                    next_state = BOARD_ERASE;
                end

                WAIT_2: begin
                    next_state = IDLE;
                end

                default: next_state = IDLE;
            endcase
        end
    endmodule
```