Xinying Wang
Yiren Zhao

Sudoku

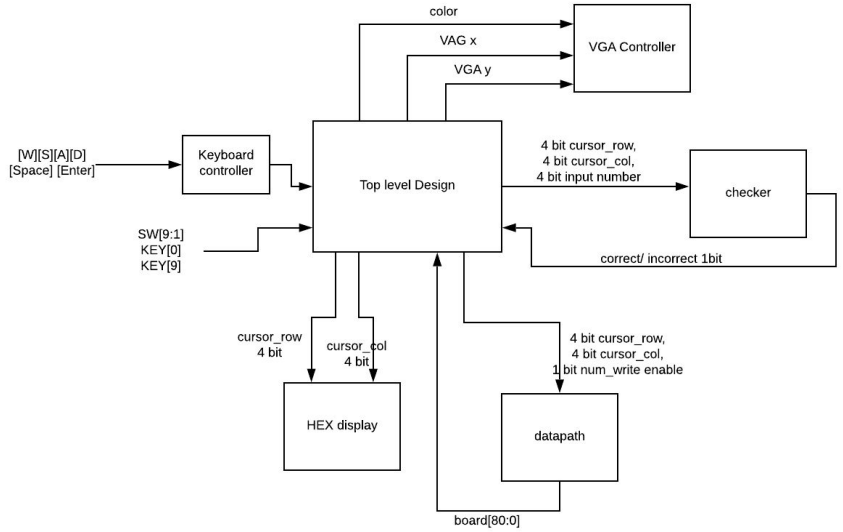
December 2, 2019

1. INTRODUCTION

As a traditional game, Sudoku has perfect mechanics and gameplay. However, when we think about sudoku, it is usually described from a software perspective, not hardware. Since our team intended to demonstrate the hardware in an unfamiliar framework, we chose to work on the down-scaled version of sudoku for the final project.

Our project combines PS2 keyboard, VGA and DE1-SoC together to improve the functionality of the game. Like most of the mini game, it has three interfaces, start page, game board and win page. To start the game, the user presses the KEY[1]. Then the game will turn into the game board page and the game would start. During the game, we mainly focus on three aspects: tracing the current location of the user, accurate position of displaying the input, as well as checking the correctness of the input and end of the game. User uses W, A, S, D on the PS2 keyboard to move to different directions. In addition, SW[1] to SW[9] control the input number, where SW[0] stands for reset all. We also generate delete functions. By pressing KEY[0], the current location with wrong input will be covered with a blank white square. Once all the boxes are filled with the correct numbers, the game will end and show a congratulation page. By pressing the KEY[2], the game will restart and change to the start interface again.

2. THE DESIGN



Block Diagram

Our core game design can be broken down into the following modules:

Top_module:

- Instantiate the VGA adapter, keyboard controller, HEX displays, checker, datapath.
- Finite state machine for plotting.

Checker:

- Contain a template board which is the solution of the game.
- Check whether each input number is correct or not.
- Output the signal to top module. It affects the rom using when user inputs the number at the current location, which vga draw the red pics if it is incorrect and draw white if it is correct.

Datapath:

- Use 81 bits register to hold the entire board.
- 1 means the position is empty, while 0 stands for the position is already occupied.
- At the start of the game, the board holds the default numbers in some position. Then each time the user input a number, the board will be updated.

Draw_borad:

- Counter for 320x120 pixels mif. Since we have multiple boards, it saves time when coding display part of project.
- Since 320x240 is larger than the maximum memory of the single rom, we divide all the board pictures into half to meet the requirement.

Draw_num:

- Counter for 18x18 pixels mif. Since we have multiple pictures of numbers(correct numbers with white background and incorrect numbers with red background), it saves time when coding display part of project.

Keyboard:

- Read the signal from the keyboard, store the signal in the register and then send the output signal to the game.
- W - move up
- A - move left
- S - move down
- D - move right

Vga:

- 320 x 240 resolution.
- Receives x,y coordinate, 6 bit colour to plot the monitor.
- Use a mif file as the game's background, which preloads the screen.

Seven_segment:

- Received signals from top level.

- It translates the current position of the user from binary numbers to human-readable number. Then display the result on HEX0 and HEX1, which are Y-coordinate and X-coordinate respectively.

Framecounter

- Create a faster clock for checker. Since the incorrect input and correct input have different patterns to display but the user can continuously change it, the checker should output the result faster than user's reactive to allow the VGA have enough time to plot the pattern.

3. REPORT ON SUCCESS

Overall, we succeeded in reaching all milestones in time and ended up with the desired result. The game ran smoothly with a clean interface through the VGA, the user can use the SW[1] to [9] to input the numbers, and use the keyboard to select the grid. The position of the selected grid displayed in HEX0 and HEX1. Also, the color of the grid were different depending on the correctness of the input number, grid with red background for incorrect input, white for correct one.

However, we encounter in the problem when we solving the current location part. Originally, we plan to draw a blue empty box(19x19 pixels with 1 pixel frame and hollow inside) around the input number to represent cursor, which moves along with user changing current location. But in fact it doesn't work as expected. The frame always shows up in unexpected grid and chaotic while erasing. To maintain the stability of the game, we decided to change the way of showing cursor by hex0 and hex1. This is also due to the time considerations because we still have other parts to work on.

The second problem appears when moving current position by pressing directional key. The cursor goes to the boundary of grid when pressing the key only once. If the current position is (4,4), it is expected to go to (4,3) if we press on up key one time. But it goes all the way up and stop at (4,0). After much time lost, we realize there is a lack of wait stage in the FSM. But at this time the deadline is approaching, we don't have time to modify this issue.

4. WHAT WOULD YOU DO DIFFERENTLY

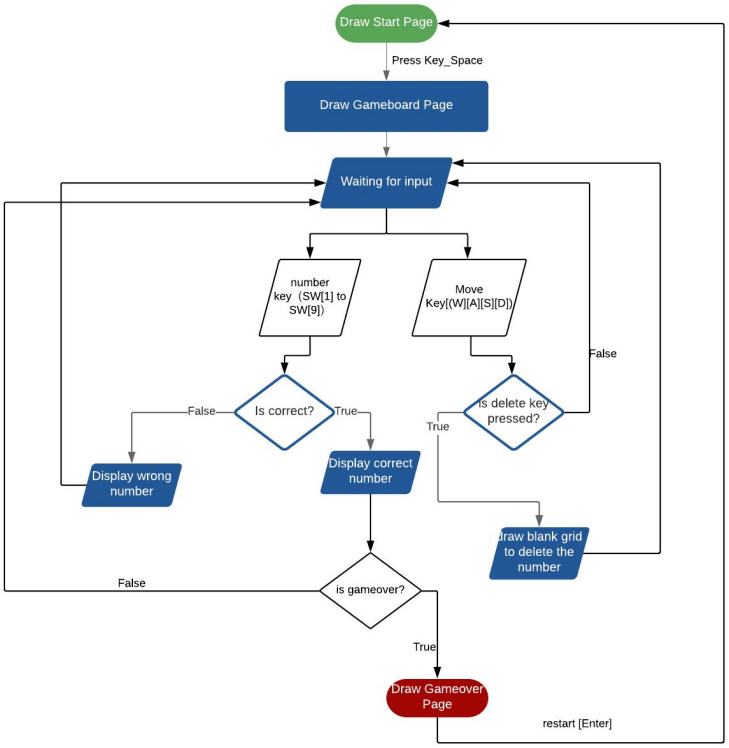
After three weeks of exploration, our team realize the importance of starting the work from the easy scope, then making it hard. At the beginning of the task, we wasted a lot of time by testing after writing the entire code. This causes that the waveform becomes too complex to analyze when we do simulation. Even in the last week, we realized that there were missing of some important stages in the FSM so that we had to overthrow the entire FSM and rewrite it again. Those experiences inspire us that we should proceed the project step by step. It will save much time when doing simulation and debugging.

Currently, we only have one game board and we typed all 81 grids by hand which took us a long time and might arise few typing errors. If we could start over again, we would consider creating a random generator to generate the gameboard which could avoid typing mistakes and make the game

has more levels, like easy, medium and hard mode. Also, we would like to add background music for the game. We were interested in implementing the audio controller using verilog, but we did not have time to apply it this time.

5. APPENDIX

FSM

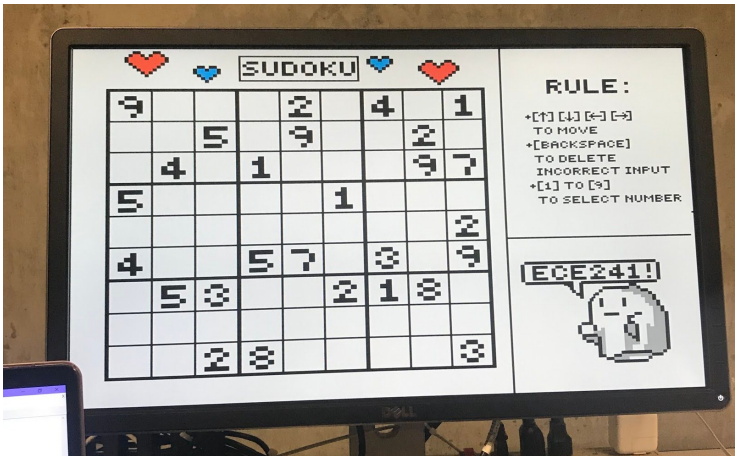


Xinying Wang
Yiren Zhao

Game Start Page

We want to do the different level at the very beginning but give it up because time limiting. So the bottom part of the screen should be “press space to start”.

Gameboard Page



Gameover Page

The bottom part of the screen should be “press enter to restart”.

Sudoku.v (top level)
Picture 1 of 19

```
module sudoku(CLOCK_50, SW,KEY,HEX0,HEX1, PS2_CLK, PS2_DAT,
              VGA_HS,VGA_VS, VGA_BLANK_N,VGA_SYNC_N,
VGA_R,VGA_G,VGA_B,VGA_CLK);

      input CLOCK_50;
      input [9:0]SW;
      input [3:0]KEY;
      output [6:0]HEX0,HEX1;

      inout PS2_CLK;
      inout PS2_DAT;

      output VGA_HS;
      output VGA_VS;
      output VGA_BLANK_N;                    //    VGA BLANK
      output VGA_SYNC_N;                     //    VGA SYNC
      output [7:0] VGA_R;                 // VGA red component
     output [7:0] VGA_G;                // VGA green component
      output [7:0] VGA_B;
      output VGA_CLK;

      wire [323:0] board_occu;
      wire isCorrect;


      wire resetn;
      assign resetn=SW[0];

      wire key_up;
      wire key_down;
      wire key_left;
      wire key_right;
      wire key_space;
      assign key_space=~KEY[1];
      wire key_delete;
      assign key_delete=~KEY[0];
      wire key_enter;
      assign key_enter=~KEY[2];


      wire key_num1;
      assign key_num1=SW[1];
      wire key_num2;
      assign key_num2=SW[2];
      wire key_num3;
      assign key_num3=SW[3];
      wire key_num4;
      assign key_num4=SW[4];
      wire key_num5;
      assign key_num5=SW[5];
      wire key_num6;
      assign key_num6=SW[6];
      wire key_num7;
      assign key_num7=SW[7];
      wire key_num8;
      assign key_num8=SW[8];
      wire key_num9;
```

Picture 2 of 19

```
	assign key_num9=SW[9];


	 keyboard k1(CLOCK_50,resetn,PS2_CLK,PS2_DAT,key_up, key_right,
key_down, key_left);



	reg [3:0] cursor_i;
	reg [3:0] cursor_j;
	reg [8:0] cursor_ij;
	reg [9:0] cursor_x;
	reg [8:0] cursor_y;
	reg [9:0] cursor_xold;
	reg [8:0] cursor_yold;

	reg [3:0]input_num;
	reg writeEn;
	reg [9:0] x;
	reg [8:0] y;
	reg [5:0]color;
	reg
isEmpty,valid_position,canDelete,deleteDone,wronDone,corrDone,gameover;
	reg checkdelete;
	reg [3:0]movecheck;
	reg num_key,move_key,delete_key;

	initial begin
		input_num = 4'd0;
		isEmpty = 1'b1;
		checkdelete = 1'd0;
		movecheck = 4'd0;
		cursor_i = 4'd4;
	  cursor_j = 4'd4;
		cursor_ij=9'd160;
		canDelete=1'b0;
		cursor_i = 4'd4;
		cursor_x= 10'd106;
		cursor_y= 9'd118;
		num_key=1'b0;
		move_key=1'b0;
		delete_key=1'b0;
		valid_position = 1'b0;
		gameover=1'b0;
		wronDone=0;
		corrDone=0;
		deleteDone=0;

		end
	wire clkout;

	frameCounter co(CLOCK_50,clkout);


	vga_adapter VGA(
		.resetn(resetn),
		.clock(CLOCK_50),
		.colour(color),
```

Picture 3 of 19

```
            .x(x),
            .y(y),
            .plot(writeEn),
            .VGA_R(VGA_R),
            .VGA_G(VGA_G),
            .VGA_B(VGA_B),
            .VGA_HS(VGA_HS),
            .VGA_VS(VGA_VS),
            .VGA_BLANK(VGA_BLANK_N),
            .VGA_SYNC(VGA_SYNC_N),
            .VGA_CLK(VGA_CLK));
      defparam VGA.RESOLUTION = "320x240";
      defparam VGA.MONOCHROME = "FALSE";
      defparam VGA.BITS_PER_COLOUR_CHANNEL = 2;
      defparam VGA.BACKGROUND_IMAGE ="test.mif";

      HexCoder YCOOR(cursor_i[3], cursor_i[2], cursor_i[1], cursor_i[0],
HEX0[6:0]);//display for y coor
      HexCoder XCOOR(cursor_j[3], cursor_j[2], cursor_j[1], cursor_j[0],
HEX1[6:0]);//display for x coor

      //--------------------draw gameboard--------------------
      wire [9:0]board_top_x, board_bot_x;
      wire [8:0]board_top_y,board_bot_y;
      wire [15:0]bTopaddress, bBotaddress;
      wire [5:0]colourboardtop, colourboardbot;
      wire boardTopDone, boardBotDone;
      reg boardTopEn,boardBotEn;

      //roms for board
      btop topback(bTopaddress, CLOCK_50, colourboardtop);
      bbot botback(bBotaddress, CLOCK_50, colourboardbot);

      //draw board(t/b)
      draw_board b1(CLOCK_50, boardTopEn, board_top_x, board_top_y,
bTopaddress, boardTopDone);
      draw_board b2(CLOCK_50, boardBotEn, board_bot_x, board_bot_y,
bBotaddress, boardBotDone);

      //--------------------draw startpage--------------------
      wire [9:0]start_top_x, start_bot_x;
      wire [8:0]start_top_y,start_bot_y;
      wire [15:0]staddress, sbaddress;
      wire [5:0]start_top_color,start_bot_color;
      wire startTopDone, startBotDone;
      reg startTopEn,startBotEn;

      //roms for start page
      starttop st(staddress, CLOCK_50, start_top_color);
      startbot sb(sbaddress, CLOCK_50, start_bot_color);

      //draw start page(t/b)
      draw_board s1(CLOCK_50, startTopEn, start_top_x, start_top_y,
staddress, startTopDone);
      draw_board s2(CLOCK_50, startBotEn, start_bot_x, start_bot_y,
sbaddress, startBotDone);
```

Picture 4 of 19

```
      //--------------------draw winpage--------------------
      wire [9:0]x_gameover1, x_gameover2;
      wire [8:0]y_gameover1, y_gameover2;
      wire [15:0]wtaddress, wbaddress;
      wire [5:0]colourgameover1, colourgameover2;
      wire gameover_top_done,gameover_bot_done;
      reg gameover1En, gameover2En;

      //rom
      wintop stttt(wtaddress, CLOCK_50, colourgameover1);
      winbot sbbbb(wbaddress, CLOCK_50, colourgameover2);

      //draw
      draw_board w1(CLOCK_50, gameover1En, x_gameover1, y_gameover1,
wtaddress, gameover_top_done);
      draw_board w2(CLOCK_50, gameover2En, x_gameover2, y_gameover2,
wbaddress, gameover_bot_done);


   reg
num1EN,num2EN,num3EN,num4EN,num5EN,num6EN,num7EN,num8EN,num9EN,deleteEN;
   reg
num1EN_in,num2EN_in,num3EN_in,num4EN_in,num5EN_in,num6EN_in,num7EN_in,num
8EN_in,num9EN_in;
  initial begin
               num1EN = 1'b0;
                 num2EN = 1'b0;
                 num3EN = 1'b0;
                 num4EN = 1'b0;
                 num5EN = 1'b0;
                 num6EN = 1'b0;
                 num7EN = 1'b0;
                 num8EN = 1'b0;
                 num9EN = 1'b0;

               num1EN_in = 1'b0;
                 num2EN_in = 1'b0;
                 num3EN_in = 1'b0;
                 num4EN_in = 1'b0;
                 num5EN_in = 1'b0;
                 num6EN_in = 1'b0;
                 num7EN_in = 1'b0;
                 num8EN_in = 1'b0;
                 num9EN_in = 1'b0;

          end

      //------------draw normal num 1-9, blank square for deleting-------
-------------
      wire [8:0] addressNum0;
      wire [8:0] num_address1;
      wire [8:0] num_address2;
      wire [8:0] num_address3;
      wire [8:0] num_address4;
      wire [8:0] num_address5;
      wire [8:0] num_address6;
      wire [8:0] num_address7;
      wire [8:0] num_address8;
```

Picture 5 of 19

```
	wire [8:0] num_address9;
	wire [5:0] num1_color,num2_color,num3_color;
	wire [5:0] num4_color,num5_color,num6_color;
	wire [5:0] num7_color,num8_color,num9_color,num0_color;
	wire [5:0] xCountNum0,yCountNum0;
	wire [5:0] xCountNum1, yCountNum1;
	wire [5:0] xCountNum2, yCountNum2;
	wire [5:0] xCountNum3, yCountNum3;
	wire [5:0] xCountNum4, yCountNum4;
	wire [5:0] xCountNum5, yCountNum5;
	wire [5:0] xCountNum6, yCountNum6;
	wire [5:0] xCountNum7, yCountNum7;
	wire [5:0] xCountNum8, yCountNum8;
	wire [5:0] xCountNum9, yCountNum9;

	wire [9:0]numDone;

	//rom
	num1 n1(num_address1, CLOCK_50, num1_color);
	num2 n2(num_address2, CLOCK_50, num2_color);
	num3 n3(num_address3, CLOCK_50, num3_color);
	num4 n4(num_address4, CLOCK_50, num4_color);
	num5 n5(num_address5, CLOCK_50, num5_color);
	num6 n6(num_address6, CLOCK_50, num6_color);
	num7 n7(num_address7, CLOCK_50, num7_color);
	num8 n8(num_address8, CLOCK_50, num8_color);
	num9 n9(num_address9, CLOCK_50, num9_color);
	blk n1110(addressNum0, CLOCK_50, num0_color);//blank box

	//draw
	drawNums numd1(CLOCK_50,  num1EN, xCountNum1, yCountNum1,
num_address1, numDone[1]);
	drawNums numd2(CLOCK_50,  num2EN, xCountNum2, yCountNum2,
num_address2, numDone[2]);
	drawNums numd3(CLOCK_50,  num3EN, xCountNum3, yCountNum3,
num_address3, numDone[3]);
	drawNums numd4(CLOCK_50,  num4EN, xCountNum4, yCountNum4,
num_address4, numDone[4]);
	drawNums numd5(CLOCK_50,  num5EN, xCountNum5, yCountNum5,
num_address5, numDone[5]);
	drawNums numd6(CLOCK_50,  num6EN, xCountNum6, yCountNum6,
num_address6, numDone[6]);
	drawNums numd7(CLOCK_50,  num7EN, xCountNum7, yCountNum7,
num_address7, numDone[7]);
	drawNums numd8(CLOCK_50,  num8EN, xCountNum8, yCountNum8,
num_address8, numDone[8]);
	drawNums numd9(CLOCK_50,  num9EN, xCountNum9, yCountNum9,
num_address9, numDone[9]);
	drawNums n0d11(CLOCK_50, deleteEN, xCountNum0, yCountNum0,
addressNum0, numDone[0]);


	//------------draw incorrect red num 1-9--------------------
	wire [8:0] num_address_incorrect1;
	wire [8:0] num_address_incorrect2;
	wire [8:0] num_address_incorrect3;
	wire [8:0] num_address_incorrect4;
	wire [8:0] num_address_incorrect5;
```

Picture 6 of 19

```
	wire [8:0] num_address_incorrect6;
	wire [8:0] num_address_incorrect7;
	wire [8:0] num_address_incorrect8;
	wire [8:0] num_address_incorrect9;
	wire [5:0] num1_color_incorrect, num2_color_incorrect,
num3_color_incorrect;
	wire [5:0] num4_color_incorrect, num5_color_incorrect,
num6_color_incorrect;
	wire [5:0] num7_color_incorrect, num8_color_incorrect,
num9_color_incorrect;

	wire [5:0] xCountNum_incorrect1, yCountNum_incorrect1;
	wire [5:0] xCountNum_incorrect2, yCountNum_incorrect2;
	wire [5:0] xCountNum_incorrect3, yCountNum_incorrect3;
	wire [5:0] xCountNum_incorrect4, yCountNum_incorrect4;
	wire [5:0] xCountNum_incorrect5, yCountNum_incorrect5;
	wire [5:0] xCountNum_incorrect6, yCountNum_incorrect6;
	wire [5:0] xCountNum_incorrect7, yCountNum_incorrect7;
	wire [5:0] xCountNum_incorrect8, yCountNum_incorrect8;
	wire [5:0] xCountNum_incorrect9, yCountNum_incorrect9;
	wire [9:0]inCorrectNumDone;
	//reg isCorrect;

	//rom
	incorrect1 wrong1(num_address_incorrect1, CLOCK_50,
num1_color_incorrect);
	incorrect2 wrong2(num_address_incorrect2, CLOCK_50,
num2_color_incorrect);
	incorrect3 wrong3(num_address_incorrect3, CLOCK_50,
num3_color_incorrect);
	incorrect4 wrong4(num_address_incorrect4, CLOCK_50,
num4_color_incorrect);
	incorrect5 wrong5(num_address_incorrect5, CLOCK_50,
num5_color_incorrect);
	incorrect6 wrong6(num_address_incorrect6, CLOCK_50,
num6_color_incorrect);
	incorrect7 wrong7(num_address_incorrect7, CLOCK_50,
num7_color_incorrect);
	incorrect8 wrong8(num_address_incorrect8, CLOCK_50,
num8_color_incorrect);
	incorrect9 wrong9(num_address_incorrect9, CLOCK_50,
num9_color_incorrect);

	//draw
	drawNums numd11(CLOCK_50, num1EN_in, xCountNum_incorrect1,
yCountNum_incorrect1, num_address_incorrect1, inCorrectNumDone[1]);
	drawNums numd12(CLOCK_50, num2EN_in, xCountNum_incorrect2,
yCountNum_incorrect2, num_address_incorrect2, inCorrectNumDone[2]);
	drawNums numd13(CLOCK_50, num3EN_in, xCountNum_incorrect3,
yCountNum_incorrect3, num_address_incorrect3, inCorrectNumDone[3]);
	drawNums numd14(CLOCK_50, num4EN_in, xCountNum_incorrect4,
yCountNum_incorrect4, num_address_incorrect4, inCorrectNumDone[4]);
	drawNums numd15(CLOCK_50, num5EN_in, xCountNum_incorrect5,
yCountNum_incorrect5, num_address_incorrect5, inCorrectNumDone[5]);
	drawNums numd16(CLOCK_50, num6EN_in, xCountNum_incorrect6,
yCountNum_incorrect6, num_address_incorrect6, inCorrectNumDone[6]);
	drawNums numd17(CLOCK_50, num7EN_in, xCountNum_incorrect7,
yCountNum_incorrect7, num_address_incorrect7, inCorrectNumDone[7]);
```

Picture 7 of 19

```
      drawNums numd18(CLOCK_50, num8EN_in, xCountNum_incorrect8,
yCountNum_incorrect8, num_address_incorrect8, inCorrectNumDone[8]);
      drawNums numd19(CLOCK_50, num9EN_in, xCountNum_incorrect9,
yCountNum_incorrect9, num_address_incorrect9, inCorrectNumDone[9]);

      reg [9:0]top_x0, bot_x0;
      reg [8:0]top_y0, bot_y0;

      initial begin
            top_x0<=0;
            top_y0<=0;
            bot_x0<=0;
            bot_y0<=9'd120;
      end

      reg [5:0] next_Draw, current_Draw;

      parameter
                  //RESET= 6'd0;
                  DISPLAY_START_TOP=6'd1,
                      DISPLAY_START_BOT=6'd2,
                      DISPLAY_BOARD_TOP=6'd4,
                      DISPLAY_BOARD_BOT=6'd5,
                      WAIT_FOR_INPUT=6'd6,
                        S_CHECK_NUM=6'd7,
                    S_CHECK_MOVE=6'd8,
                    S_CHECK_DELETE=6'd9,
                    S_CHECK_NUM_FEEDBACK=6'd10,
                  S_CHECK_MOVE_FEEDBACK=6'd11,
                    S_CHECK_DELETE_FEEDBACK=6'd12,
                        DISPLAY_NUM_WAIT=6'd13,
                    DISPLAY_NUM=6'd14,
                    UPDATE_POSITION=6'd15,
                    DELETE_NUM=6'd16,
                    DISPLAY_WRON=6'd17,
                    DISPLAY_CORRECT=6'd18,
                    CHECK_GAMEOVER=6'd19,
                      GAMEOVER_TOP=6'd20,
                      GAMEOVER_BOT=6'd21;


      always@(*)
      begin
      case(current_Draw)
           /*     RESET:      begin
                               next_Draw <=DISPLAY_START_TOP;
                               end*/

            DISPLAY_START_TOP:begin
                                    if (startTopEn) begin
                                         if (startTopDone) begin
                                               next_Draw <=
DISPLAY_START_BOT;
                                         end
                                         else next_Draw <=
DISPLAY_START_TOP;
                                         end
                                    end
```

Picture 8 of 19

```
            DISPLAY_START_BOT:      begin
                                     if (startBotEn) begin
                                          if (startBotDone) begin
                                               if(key_space)
                                                    next_Draw <=
DISPLAY_BOARD_TOP;
                                          end
                                          else next_Draw <=
DISPLAY_START_BOT;
                                          end
                                    end

            DISPLAY_BOARD_TOP:      begin
                                     if (boardTopEn) begin
                                          if (boardTopDone) begin
                                               next_Draw <=
DISPLAY_BOARD_BOT;
                                          end
                                          else next_Draw <=
DISPLAY_BOARD_TOP;
                                          end
                                    end
            DISPLAY_BOARD_BOT:      begin
                                     if (boardBotEn) begin
                                          if (boardBotDone) begin
                                               next_Draw <=
WAIT_FOR_INPUT;
                                          end
                                          else next_Draw <=
DISPLAY_BOARD_BOT;
                                          end
                                    end

            WAIT_FOR_INPUT: begin
                            if(input_num != 4'd0)
                                    next_Draw <= S_CHECK_NUM;
                            else if(movecheck != 4'd0)
                                    next_Draw <= S_CHECK_MOVE;
                            else if(checkdelete)
                                   next_Draw <= S_CHECK_DELETE;
                            else next_Draw <= WAIT_FOR_INPUT;
                            end

            S_CHECK_NUM:  next_Draw <= S_CHECK_NUM_FEEDBACK;
            S_CHECK_MOVE: next_Draw <= S_CHECK_MOVE_FEEDBACK;
            S_CHECK_DELETE: next_Draw <= S_CHECK_DELETE_FEEDBACK;

            S_CHECK_NUM_FEEDBACK:next_Draw <= (board_occu[cursor_ij] ==1
&& board_wrong[cursor_ij] ==1) ? DISPLAY_NUM_WAIT : WAIT_FOR_INPUT;

            S_CHECK_MOVE_FEEDBACK:  begin
                                    if(~valid_position)
                                                next_Draw <=
WAIT_FOR_INPUT ;
                                            else next_Draw <=
UPDATE_POSITION;
                                            end
```

Picture 9 of 19

```
            S_CHECK_DELETE_FEEDBACK: next_Draw <=
(board_wrong[cursor_ij]==1'b0 && board_occu[cursor_ij]==1'b1) ?
DELETE_NUM : WAIT_FOR_INPUT;
            DISPLAY_NUM_WAIT:next_Draw <= DISPLAY_NUM;
            DISPLAY_NUM: next_Draw <= isCorrect ? DISPLAY_CORRECT :
DISPLAY_WRON ;
            UPDATE_POSITION: next_Draw <= WAIT_FOR_INPUT;
            DELETE_NUM: next_Draw <= deleteDone ? WAIT_FOR_INPUT :
DELETE_NUM ;

            DISPLAY_WRON: next_Draw <= wronDone ? WAIT_FOR_INPUT :
DISPLAY_WRON;
            DISPLAY_CORRECT:next_Draw <= corrDone ?  CHECK_GAMEOVER :
DISPLAY_CORRECT;

            CHECK_GAMEOVER:next_Draw <= (board_occu==81'b0) ?
GAMEOVER_TOP :  WAIT_FOR_INPUT;

            GAMEOVER_TOP:     begin
                                    if (gameover1En) begin
                                          if (gameover_top_done) begin
                                                next_Draw <= GAMEOVER_BOT;
                                                end
                                          else next_Draw <= GAMEOVER_TOP;
                                          end
                                    end
            GAMEOVER_BOT:     begin
                                    if (gameover2En) begin
                                          if (gameover_bot_done) begin
                                                if(key_enter)
                                                      next_Draw <=
DISPLAY_START_TOP;
                                                end
                                          else next_Draw <= GAMEOVER_BOT;
                                          end
                                    end
            default: next_Draw<=DISPLAY_START_TOP;
            endcase
      end


      //stageEn for boards

      always @ (posedge CLOCK_50)
      begin
            if (current_Draw==DISPLAY_START_TOP) begin
                  startTopEn <= 1;
                  end
            else startTopEn <= 0;
      end

      always @ (posedge CLOCK_50)
      begin
            if (current_Draw==DISPLAY_START_BOT) begin
                  startBotEn <= 1;
                  end
            else startBotEn <= 0;
      end
```

Picture 10 of 19

```
	always @ (posedge CLOCK_50)
	begin
		if (current_Draw==DISPLAY_BOARD_TOP) begin
			boardTopEn <= 1;
			end
		else boardTopEn <= 0;
	end

	always @ (posedge CLOCK_50)
	begin
		if (current_Draw==DISPLAY_BOARD_BOT) begin
			boardBotEn <= 1;
			end
		else boardBotEn <= 0;
	end


	always @ (posedge CLOCK_50)
	begin
		if (current_Draw==GAMEOVER_TOP) begin
			gameover1En <= 1;
			end
		else gameover1En <= 0;
	end

	always @ (posedge CLOCK_50)
	begin
		if (current_Draw==GAMEOVER_BOT) begin
			gameover2En <= 1;
			end
		else gameover2En <= 0;
	end


	always @ (posedge CLOCK_50 or negedge resetn)		// Shifting
states
	begin
		if (resetn==0)
			current_Draw <= DISPLAY_START_TOP;
		else
			current_Draw <= next_Draw;
	end

	always@(*) begin

		if (resetn==0)begin
		cursor_i = 4'd4;
	  cursor_j = 4'd4;
		cursor_ij=9'd40;
		cursor_x= 10'd106;
		cursor_y= 9'd118;
		cursor_xold = 10'd105;
		cursor_yold = 9'd118;
		input_num = 4'd0;
		end

		else if (current_Draw == WAIT_FOR_INPUT) begin
```

Picture 11 of 19

```
                if(key_num1)
                      input_num=4'd1;
                else if (key_num2)
                      input_num=4'd2;
                else if (key_num3)
                      input_num=4'd3;
                else if (key_num4)
                      input_num=4'd4;
                else if (key_num5)
                      input_num=4'd5;
                else if (key_num6)
                      input_num=4'd6;
                else if (key_num7)
                      input_num=4'd7;
                else if (key_num8)
                      input_num=4'd8;
                else if (key_num9)
                      input_num=4'd9;
                else if(key_delete)
                   checkdelete=1'b1;
                else if(key_up)
                movecheck = 4'd1;
                else if(key_down)
                movecheck = 4'd2;
                else if(key_left)
                movecheck = 4'd3;
                else if(key_right)
                movecheck = 4'd4;

          end

          else if (current_Draw== UPDATE_POSITION) begin

                if(movecheck == 4'd1)begin
                      cursor_i = cursor_i - 4'd1;
                      cursor_ij= cursor_ij - 9'd9;
                      cursor_x= cursor_x;
                      cursor_y= cursor_y - 9'd21;
                      movecheck = 0;

                end
                else if (movecheck == 4'd2) begin

                      cursor_i = cursor_i + 4'd1;
                      cursor_ij= cursor_ij + 9'd9;
                      cursor_x= cursor_x;
                      cursor_y= cursor_y + 9'd21;
                            movecheck = 0;

                end
                else if (movecheck == 4'd3) begin

                      cursor_j = cursor_j - 4'd1;
                   cursor_ij= cursor_ij - 9'd1;
                   cursor_x= cursor_x - 10'd21;
                   cursor_y= cursor_y;
                            movecheck = 0;
```

Picture 12 of 19

```
                    end
                    else if (movecheck == 4'd3 ) begin

                          cursor_j = cursor_j + 4'd1;
                          cursor_ij= cursor_ij + 9â€˜d1;
                          cursor_x= cursor_x + 10'd21;
                          cursor_y= cursor_y;
                                movecheck = 0;
                    end
              end
        end


        always @ (posedge CLOCK_50)
        begin

              if (resetn==0)
              begin
                  checkdelete = 1'd0;
                  movecheck = 4'd0;
                  canDelete=1'b0;
                  num_key=1'b0;
                  move_key=1'b0;
                  delete_key=1'b0;
                  valid_position = 1'b0;
                    gameover=1'b0;
                    wronDone=0;
                    corrDone=0;
                    deleteDone=0;
              end

      //draw startpage
              if (current_Draw == DISPLAY_START_TOP) begin
                    if (startTopEn) begin
                          x <= top_x0+ start_top_x;
                          y <= top_y0+ start_top_y;
                          color <= start_top_color;
                          writeEn <= startTopEn;
                    end
              end

              else if (current_Draw == DISPLAY_START_BOT) begin
                    if (startBotEn) begin
                          x <= bot_x0+ start_bot_x;
                          y <= bot_y0+start_bot_y;
                          color <= start_bot_color;
                          writeEn <= startBotEn;

                    end
              end

              //draw gameboard
              else if (current_Draw == DISPLAY_BOARD_TOP) begin
                    if (boardTopEn) begin
                          x <= top_x0+ board_top_x;
                          y <= top_y0+ board_top_y;
                          color <= colourboardtop;
                          writeEn <= boardTopEn;
```

Picture 13 of 19

```
      end
end

else if (current_Draw == DISPLAY_BOARD_BOT) begin
      if (boardBotEn) begin
            x <= bot_x0+ board_bot_x;
            y <= bot_y0+board_bot_y;
            color <= colourboardbot;
            writeEn <= boardBotEn;
      end
end

//draw winpage
else if (current_Draw == GAMEOVER_TOP) begin
      if (gameover1En) begin
            x <= top_x0+x_gameover1;
            y <= top_y0+y_gameover1;
            color <= colourgameover1;
            writeEn <= gameover1En;
      end
end

else if (current_Draw == GAMEOVER_BOT) begin
      if (gameover2En) begin
            x <= bot_x0+x_gameover2;
            y <= bot_y0+y_gameover2;
            color <= colourgameover2;
            writeEn <= gameover2En;
      end
end




else if(current_Draw==DISPLAY_NUM_WAIT)begin
if(isCorrect)begin
    if(input_num==4'd1) num1EN = 1'b1;
      if(input_num==4'd2) num2EN = 1'b1;
      if(input_num==4'd3) num3EN = 1'b1;
      if(input_num==4'd4) num4EN = 1'b1;
      if(input_num==4'd5) num5EN = 1'b1;
      if(input_num==4'd6) num6EN = 1'b1;
      if(input_num==4'd7) num7EN = 1'b1;
      if(input_num==4'd8) num8EN = 1'b1;
      if(input_num==4'd9) num9EN = 1'b1;
end
else if(!isCorrect)begin
    if(input_num==4'd1) num1EN_in = 1'b1;
      if(input_num==4'd2) num2EN_in = 1'b1;
      if(input_num==4'd3) num3EN_in = 1'b1;
      if(input_num==4'd4) num4EN_in = 1'b1;
      if(input_num==4'd5) num5EN_in = 1'b1;
      if(input_num==4'd6) num6EN_in = 1'b1;
      if(input_num==4'd7) num7EN_in = 1'b1;
      if(input_num==4'd8) num8EN_in = 1'b1;
      if(input_num==4'd9) num9EN_in = 1'b1;
end
```

Picture 14 of 19

```
            end

            /*else if (current_Draw == S_CHECK_NUM) begin
                if((board_occu[cursor_ij] == 0) &&
(board_occu[cursor_ij+9'd1] == 0 )&& (board_occu[cursor_ij+9'd2] ==0)  &&
(board_occu[cursor_ij+9'd3] == 0) )
               isEmpty = 1'b1;
                  else isEmpty = 1'b0;
            end
            */

            else if (current_Draw== S_CHECK_MOVE) begin
                              if(((movecheck ==4'd1) && cursor_i >0 ) ||
((movecheck == 4'd2) && cursor_i < 4'd8) ||
                              ((movecheck == 4'd3) && cursor_j >0)  ||
((movecheck == 4'd4) && cursor_j < 4'd8))
                                    valid_position = 1'b1;
                              else valid_position = 1'b0;
                              end

            else if (current_Draw== S_CHECK_DELETE) begin
                  //if(~isCorrect) begin
                              canDelete=1'b1;
                              deleteEN = 1'b1;
                           // end
                           //else canDelete=1'b0;
            end



            else if (current_Draw==DISPLAY_CORRECT) begin

                     if(input_num == 4'd1)begin
                              x <= cursor_x+ xCountNum1;
                           y <= cursor_y+ yCountNum1;
                           color <= num1_color;
                              corrDone <= 0;
                           if(numDone[1]) begin
                              corrDone <= 1;
                                 input_num = 4'd0;
                                 num1EN = 1'b0;
                              end
                              end

                      else if(input_num == 4'd2)begin
                              x <= cursor_x+ xCountNum2;
                           y <= cursor_y+ yCountNum2;
                           color <= num2_color;
                              corrDone <= 0;

                              if(numDone[2])
                              begin
                              corrDone <= 1;
                               input_num = 4'd0;
                                 num2EN = 1'b0;
                              end
                              end
```

Picture 15 of 19

```
  else if(input_num == 4'd3)begin
         x <= cursor_x+ xCountNum3;
       y <= cursor_y+ yCountNum3;
       color <= num3_color;
         corrDone <= 0;

         if(numDone[3])
         begin
         corrDone <= 1;
          input_num = 4'd0;
            num3EN = 1'b0;
         end
         end

  else if(input_num == 4'd4)begin
         x <= cursor_x+ xCountNum4;
       y <= cursor_y+ yCountNum4;
       color <= num4_color;
         corrDone <= 0;

         if(numDone[4])
         begin
         corrDone <= 1;
          input_num = 4'd0;
            num4EN = 1'b0;
         end
         end

else if(input_num == 4'd5)begin
         x <= cursor_x+ xCountNum5;
       y <= cursor_y+ yCountNum5;
       color <= num5_color;
         corrDone <= 0;

         if(numDone[5])
         begin
         corrDone <= 1;
          input_num = 4'd0;
            num5EN = 1'b0;
         end
         end

 else if(input_num == 4'd6)begin
         x <= cursor_x+ xCountNum6;
       y <= cursor_y+ yCountNum6;
       color <= num6_color;
         corrDone <= 0;

         if(numDone[6])
         begin
         corrDone <= 1;
          input_num = 4'd0;
            num6EN = 1'b0;
         end
         end

  else if(input_num == 4'd7)begin
         x <= cursor_x+ xCountNum7;
```

Picture 16 of 19

```
                y <= cursor_y+ yCountNum7;
                color <= num7_color;
                   corrDone <= 0;

                   if(numDone[7])
                   begin
                   corrDone <= 1;
                    input_num = 4'd0;
                      num7EN = 1'b0;
                   end
                   end

             else if(input_num == 4'd8)begin
                      x <= cursor_x+ xCountNum8;
                y <= cursor_y+ yCountNum8;
                color <= num8_color;
                   corrDone <= 0;

                   if(numDone[8])
                   begin
                   corrDone <= 1;
                    input_num = 4'd0;
                      num8EN = 1'b0;
                   end
                   end

            else if(input_num == 4'd9)begin
                      x <= cursor_x+ xCountNum9;
                y <= cursor_y+ yCountNum9;
                color <= num9_color;
                   corrDone <= 0;

                   if(numDone[9])
                   begin
                   corrDone <= 1;
                    input_num = 4'd0;
                      num9EN = 1'b0;
                   end
                   end

end

else if (current_Draw==DISPLAY_WRON) begin

            if(input_num == 4'd1)begin
                      x <= cursor_x+ xCountNum_incorrect1;
                y <= cursor_y+ yCountNum_incorrect1;
                color <= num1_color_incorrect;
                   wronDone <= 0;
                   if(inCorrectNumDone[1]) begin
                   wronDone <= 1;
                      input_num = 4'd0;
                      num1EN_in = 1'b0;
                   end
                   end

             else if(input_num == 4'd2)begin
                      x <= cursor_x+ xCountNum_incorrect2;
```

Picture 17 of 19

```
        y <= cursor_y+ yCountNum_incorrect2;
       color <= num2_color_incorrect;
               wronDone <= 0;
          if(inCorrectNumDone[2]) begin
          wronDone <= 1;
             input_num = 4'd0;
             num2EN_in = 1'b0;
          end
          end

    else if(input_num == 4'd3)begin
          x <= cursor_x+ xCountNum_incorrect3;
        y <= cursor_y+ yCountNum_incorrect3;
       color <= num3_color_incorrect;
               wronDone <= 0;
          if(inCorrectNumDone[3]) begin
          wronDone <= 1;
             input_num = 4'd0;
             num3EN_in = 1'b0;
          end
          end

    else if(input_num == 4'd4)begin
          x <= cursor_x+ xCountNum_incorrect4;
       y <= cursor_y+ yCountNum_incorrect4;
       color <= num4_color_incorrect;
               wronDone <= 0;
          if(inCorrectNumDone[4]) begin
          wronDone <= 1;
             input_num = 4'd0;
             num4EN_in = 1'b0;
          end
          end

  else if(input_num == 4'd5)begin
          x <= cursor_x+ xCountNum_incorrect5;
       y <= cursor_y+ yCountNum_incorrect5;
       color <= num5_color_incorrect;
               wronDone <= 0;
          if(inCorrectNumDone[5]) begin
          wronDone <= 1;
             input_num = 4'd0;
             num5EN_in = 1'b0;
          end
          end

   else if(input_num == 4'd6)begin
          x <= cursor_x+ xCountNum_incorrect6;
       y <= cursor_y+ yCountNum_incorrect6;
       color <= num6_color_incorrect;
               wronDone <= 0;
          if(inCorrectNumDone[6]) begin
          wronDone <= 1;
             input_num = 4'd0;
             num6EN_in = 1'b0;
          end
          end
```

Picture 18 of 19

```
                    else if(input_num == 4'd7)begin
                           x <= cursor_x+ xCountNum_incorrect7;
                         y <= cursor_y+ yCountNum_incorrect7;
                         color <= num7_color_incorrect;
                                 wronDone <= 0;
                           if(inCorrectNumDone[7]) begin
                           wronDone <= 1;
                              input_num = 4'd0;
                              num7EN_in = 1'b0;
                           end
                           end

                    else if(input_num == 4'd8)begin
                           x <= cursor_x+ xCountNum_incorrect8;
                         y <= cursor_y+ yCountNum_incorrect8;
                         color <= num8_color_incorrect;
                                 wronDone = 0;
                           if(inCorrectNumDone[8]) begin
                           wronDone <= 1;
                              input_num = 4'd0;
                              num8EN_in = 1'b0;
                           end
                           end

                   else if(input_num == 4'd9)begin
                           x <= cursor_x+ xCountNum_incorrect9;
                         y <= cursor_y+ yCountNum_incorrect9;
                         color <= num9_color_incorrect;
                           wronDone <= 0;
                           if(inCorrectNumDone[9]) begin
                           wronDone <= 1;
                              input_num = 4'd0;
                              num9EN_in = 1'b0;
                           end
                           end
             end

              else if (current_Draw==DELETE_NUM) begin
                     x <= cursor_x+ xCountNum0;
                        y <= cursor_y+ yCountNum0;
                        color <= 6'b111111;
                      deleteDone <= 0;

                        if(numDone[0]) begin
                        deleteDone<=1'b0;
                        checkdelete = 4'd0;
                        deleteEN = 1'b0;
                        end
              end

                else if (current_Draw==CHECK_GAMEOVER) begin
        end
end


     checker check(CLOCK_50, resetn, input_num, cursor_i,cursor_j,
isCorrect);
```

Picture 19 of 19

```
        sudoku_datapath path(CLOCK_50,resetn,
    isCorrect,cursor_ij,board_occu);


    endmodule
```

Draw_board.v

```
module draw_board (clk, enable, xCount, yCount, address, done);
    //draw 320x120 board

    input clk, enable;
    output reg [9:0]xCount;
    output reg [8:0]yCount;
    output reg [15:0]address;
    output reg done;

    initial begin
    xCount = 0;
    yCount = 0;
    address = 0;
    end

    always @ (posedge clk)
    begin
    if (enable) begin
        if (xCount < 10'd319)
                xCount <= xCount +1;
         else if (xCount==10'd319) begin
            if (yCount<9'd119) begin
                xCount<=0;
                yCount<=yCount+1;
                end
            else if (yCount==9'd119) done<=1;
            end
        address <= address+16'b1;
        end
    else begin
        done <= 0;
        address <= 2;
        xCount = 0;
        yCount = 0;
        end
    end

 endmodule
```

drawNums.v

```
module drawNums(clk, enable, xCount, yCount, address, done);
//18x18
//num1 to num9
    input clk, enable;
    output  reg [5:0]xCount;
    output  reg [5:0]yCount;
    output reg [8:0]address;
    output reg done;

    initial begin
    xCount = 0;
    yCount = 0;
    address = 2;
    end

    always @ (posedge clk)
    begin
    if (enable) begin
        if (xCount < 6'd17)
                xCount <= xCount +1;

        else if (xCount==6'd17) begin
            if (yCount<6'd17) begin
                xCount<=0;
                yCount<=yCount+1;
            end
            else if (yCount==6'd17) done<=1;
            end
        address <= address+9'b1;
        end
        else begin
        done <= 0;
        xCount = 0;
        yCount = 0;
        address <= 2;
        end
    end
endmodule
```

Framecounter.v

```
module frameCounter(clkin,clkout);
    input clkin;
    output reg clkout = 0;
    reg [26:0] counter = 0;
    always @(posedge clkin)
    begin
        if (counter == 0)
        begin
            counter <= (50000000/5-1);
            clkout <= 1;
        end
        else
        begin
            counter <= counter -1;
            clkout <= 0;
        end
    end
endmodule
```

Checker.v

```
module checker(clk, clr, input_num, input_row, input_col, isCorrect);
    input clk;
    input clr;
    input [3:0]input_num;
    input [3:0]input_row;
    input [3:0]input_col;
    output reg isCorrect;
    wire [3:0]template_grid[8:0][8:0];
    assign template_grid[0][0] = 4'd9;
    assign template_grid[0][1] = 4'd8;
    assign template_grid[0][2] = 4'd7;
    assign template_grid[0][3] = 4'd6;
    assign template_grid[0][4] = 4'd2;
    assign template_grid[0][5] = 4'd5;
    assign template_grid[0][6] = 4'd4;
    assign template_grid[0][7] = 4'd3;
    assign template_grid[0][8] = 4'd1;
    assign template_grid[1][0] = 4'd3;
    assign template_grid[1][1] = 4'd1;
    assign template_grid[1][2] = 4'd5;
    assign template_grid[1][3] = 4'd4;
    assign template_grid[1][4] = 4'd9;
    assign template_grid[1][5] = 4'd7;
    assign template_grid[1][6] = 4'd6;
    assign template_grid[1][7] = 4'd2;
    assign template_grid[1][8] = 4'd8;
    assign template_grid[2][0] = 4'd2;
    assign template_grid[2][1] = 4'd4;
    assign template_grid[2][2] = 4'd6;
    assign template_grid[2][3] = 4'd1;
    assign template_grid[2][4] = 4'd3;
    assign template_grid[2][5] = 4'd8;
    assign template_grid[2][6] = 4'd5;
    assign template_grid[2][7] = 4'd9;
    assign template_grid[2][8] = 4'd7;
    assign template_grid[3][0] = 4'd5;
    assign template_grid[3][1] = 4'd3;
    assign template_grid[3][2] = 4'd9;
    assign template_grid[3][3] = 4'd2;
    assign template_grid[3][4] = 4'd8;
    assign template_grid[3][5] = 4'd1;
    assign template_grid[3][6] = 4'd7;
    assign template_grid[3][7] = 4'd4;
    assign template_grid[3][8] = 4'd6;
    assign template_grid[4][0] = 4'd6;
    assign template_grid[4][1] = 4'd7;
    assign template_grid[4][2] = 4'd1;
    assign template_grid[4][3] = 4'd3;
    assign template_grid[4][4] = 4'd4;
    assign template_grid[4][5] = 4'd9;
    assign template_grid[4][6] = 4'd8;
    assign template_grid[4][7] = 4'd5;
    assign template_grid[4][8] = 4'd2;
    assign template_grid[5][0] = 4'd4;
    assign template_grid[5][1] = 4'd2;
    assign template_grid[5][2] = 4'd8;
```

```
    assign template_grid[5][3] = 4'd5;
    assign template_grid[5][4] = 4'd7;
    assign template_grid[5][5] = 4'd6;
    assign template_grid[5][6] = 4'd3;
    assign template_grid[5][7] = 4'd1;
    assign template_grid[5][8] = 4'd9;
    assign template_grid[6][0] = 4'd7;
    assign template_grid[6][1] = 4'd5;
    assign template_grid[6][2] = 4'd3;
    assign template_grid[6][3] = 4'd9;
    assign template_grid[6][4] = 4'd6;
    assign template_grid[6][5] = 4'd2;
    assign template_grid[6][6] = 4'd1;
    assign template_grid[6][7] = 4'd8;
    assign template_grid[6][8] = 4'd4;
    assign template_grid[7][0] = 4'd8;
    assign template_grid[7][1] = 4'd9;
    assign template_grid[7][2] = 4'd4;
    assign template_grid[7][3] = 4'd7;
    assign template_grid[7][4] = 4'd1;
    assign template_grid[7][5] = 4'd3;
    assign template_grid[7][6] = 4'd2;
    assign template_grid[7][7] = 4'd6;
    assign template_grid[7][8] = 4'd5;
    assign template_grid[8][0] = 4'd1;
    assign template_grid[8][1] = 4'd6;
    assign template_grid[8][2] = 4'd2;
    assign template_grid[8][3] = 4'd8;
    assign template_grid[8][4] = 4'd5;
    assign template_grid[8][5] = 4'd4;
    assign template_grid[8][6] = 4'd9;
    assign template_grid[8][7] = 4'd7;
    assign template_grid[8][8] = 4'd3;

    //input [3:0]input_grid[8:0][8:0];
    //template answer(template_grid);

    always@(posedge clk) begin
        if (!clr) begin
            isCorrect = 0;

        end
        else if(input_num==template_grid[input_row][input_col])
            isCorrect = 1'b1;
            else
            isCorrect = 1'b0;

    end
endmodule
```

Datapath.v

```
module sudoku_datapath(clk,rst,write,write_ij,board_out);
   input clk;
   input rst;
   input write; //enable-write signal
   input [8:0] write_ij;
   output [80:0] board_out ;        //row info for logic
   reg [80:0] board_occu;

    always @ (posedge clk, negedge rst) begin
        if (!rst ) begin
                board_occu[0] <= 1'd0;
                board_occu[1] <= 1'd1;
                board_occu[2] <= 1'd1;
                board_occu[3] <= 1'd1;
                board_occu[4] <= 1'd0;
                board_occu[5] <= 1'd1;
                board_occu[6] <= 1'd0;
                board_occu[7] <= 1'd1;
                board_occu[8] <= 1'd0;

                board_occu[9] <= 1'd1;
                board_occu[10] <= 1'd1;
                board_occu[11] <= 1'd0;
                board_occu[12] <= 1'd1;
                board_occu[13] <= 1'd0;
                board_occu[14] <= 1'd1;
                board_occu[15] <= 1'd1;
                board_occu[16] <= 1'd0;
                board_occu[17] <= 1'd1;

                board_occu[18] <= 1'd1;
                board_occu[19] <= 1'd0;
                board_occu[20] <= 1'd1;
                board_occu[21] <= 1'd0;
                board_occu[22] <= 1'd1;
                board_occu[23] <= 1'd1;
                board_occu[24] <= 1'd1;
                board_occu[25] <= 1'd0;
                board_occu[26] <= 1'd0;

                board_occu[27] <= 1'd0;
                board_occu[28] <= 1'd1;
                board_occu[29] <= 1'd1;
                board_occu[30] <= 1'd1;
                board_occu[31] <= 1'd1;
                board_occu[32] <= 1'd0;
                board_occu[33] <= 1'd1;
                board_occu[34] <= 1'd1;
                board_occu[35] <= 1'd1;
```

```
                board_occu[36] <= 1'd1;
                board_occu[37] <= 1'd1;
                board_occu[38] <= 1'd1;
                board_occu[39] <= 1'd1;
                board_occu[40] <= 1'd1;
                board_occu[41] <= 1'd1;
                board_occu[42] <= 1'd1;
                board_occu[43] <= 1'd1;
                board_occu[44] <= 1'd0;

                board_occu[45] <= 1'd0;
                board_occu[46] <= 1'd1;
                board_occu[47] <= 1'd1;
                board_occu[48] <= 1'd0;
                board_occu[49] <= 1'd0;
                board_occu[50] <= 1'd1;
                board_occu[51] <= 1'd0;
                board_occu[52] <= 1'd1;
                board_occu[53] <= 1'd0;

                board_occu[54] <= 1'd1;
                board_occu[55] <= 1'd0;
                board_occu[56] <= 1'd0;
                board_occu[57] <= 1'd1;
                board_occu[58] <= 1'd1;
                board_occu[59] <= 1'd0;
                board_occu[60] <= 1'd0;
                board_occu[61] <= 1'd0;
                board_occu[62] <= 1'd1;

                board_occu[63] <= 1'd1;
                board_occu[64] <= 1'd1;
                board_occu[65] <= 1'd1;
                board_occu[66] <= 1'd1;
                board_occu[67] <= 1'd1;
                board_occu[68] <= 1'd1;
                board_occu[69] <= 1'd1;
                board_occu[70] <= 1'd1;
                board_occu[71] <= 1'd1;

                board_occu[72] <= 1'd1;
                board_occu[73] <= 1'd1;
                board_occu[74] <= 1'd0;
                board_occu[75] <= 1'd0;
                board_occu[76] <= 1'd1;
                board_occu[77] <= 1'd1;
                board_occu[78] <= 1'd1;
                board_occu[79] <= 1'd1;
                board_occu[80] <= 1'd0;

            end
                end
        else if (write)begin
            //board_default[write_i][write_j] <= write_num;
                board_occu[write_ij] <= 1'b0;
                end
    end
     assign board_out=board_occu[80:0];
endmodule
```

Hexcoder.v

```
module HexCoder (input c3, c2, c1, c0, output [6:0] HEX);
    assign HEX[0] = (~c3 & ~c2 & ~c1 & c0) | (~c3 & c2 & ~c1 & ~c0) | (c3 & ~c2 & c1 & c0) | (c3 & c2 & ~c1 & c0);
    assign HEX[1] = (~c3 & c2 & ~c1 & c0) | (~c3 & c2 & c1 & ~c0) | (c3 & ~c2 & c1 & c0) | (c3& c2 & ~c1 & ~c0) |(c3 & c2 & c1 & ~c0) | (c3 & c2 & c1 & c0);
    assign HEX[2] = (~c3 & ~c2 & c1 & ~c0) | (c3 & c2 & ~c1 & ~c0) | (c3 & c2 & c1 & ~c0) | (c3 & c2 & c1 & c0);
    assign HEX[3] = (~c3 & ~c2 & ~c1 & c0) | (~c3 & c2 & ~c1 & ~c0) | (~c3 & c2 & c1 & c0) | (c3 & ~c2 & c1 & ~c0) | (c3 & c2 & c1 & c0);
    assign HEX[4] = (~c3 & ~c2 & ~c1 & c0) | (~c3 & ~c2 & c1 & c0) | (~c3 & c2 & ~c1 & ~c0) | (~c3 & c2 & ~c1 & c0) | (~c3 & c2 & c1 & c0) | (c3 & ~c2 & ~c1 & c0);
    assign HEX[5] = (~c3 & ~c2 & ~c1 & c0) | (~c3 & ~c2 & c1 & ~c0) | (~c3 & ~c2 & c1 & c0) | (~c3 & c2 & c1 & c0) | (c3 & c2 & ~c1 & c0);
    assign HEX[6] = (~c3 & ~c2 & ~c1 & ~c0) | (~c3 & ~c2 & ~c1 & c0) | (~c3 & c2 & c1 & c0) | (c3 & c2 & ~c1 & ~c0);
endmodule
```

Keyboard.v

```
module keyboard(CLOCK_50, PS2_CLK, PS2_DAT, reset, up, down, left, right);
    input CLOCK_50,reset;
    inout PS2_CLK, PS2_DAT;
    output reg up, down, left, right;
    wire        [7:0]   ps2_key_data;
    wire                ps2_key_pressed;
    reg [7:0] ps2_key_data_1, ps2_key_data_2, ps2_key_data_3;//regs
    //dataflow
    always @(posedge CLOCK_50) begin
        if (!reset) begin
            ps2_key_data_1 <= 8'b0;
            ps2_key_data_2 <= 8'b0;
            ps2_key_data_3 <= 8'b0;
        end
        else if (ps2_key_pressed) begin
            ps2_key_data_1 <= ps2_key_data_2;
            ps2_key_data_2 <= ps2_key_data_3;
            ps2_key_data_3 <= ps2_key_data;
        end
    end
    //up(w)
    always @(posedge CLOCK_50) begin
        if (!reset)
            up <= 1'b0;
        else if (ps2_key_data_2 == 8'hE0 && ps2_key_data_3 == 8'h1d)
            up <= 1'b1;
        else if (ps2_key_data_1 == 8'hE0 && ps2_key_data_2 == 8'hF0 && ps2_key_data_3 == 8'h1d)
            up <= 1'b0;
    end
    //down(s)
    always @(posedge CLOCK_50) begin
        if (!reset)
            down <= 1'b0;
        else if (ps2_key_data_2 == 8'hE0 && ps2_key_data_3 == 8'h1b)
            down <= 1'b1;
        else if (ps2_key_data_1 == 8'hE0 && ps2_key_data_2 == 8'hF0 && ps2_key_data_3 == 8'h1b)
            down <= 1'b0;
    end
    //left(a)
    always @(posedge CLOCK_50) begin
        if (!reset)
            left <= 1'b0;
        else if (ps2_key_data_2 == 8'hE0 && ps2_key_data_3 == 8'h1c)
            left <= 1'b1;
        else if (ps2_key_data_1 == 8'hE0 && ps2_key_data_2 == 8'hF0 && ps2_key_data_3 == 8'h1c)
            left <= 1'b0;
    end
    //right(d)
    always @(posedge CLOCK_50) begin
        if (!reset)
            right <= 1'b0;
        else if (ps2_key_data_2 == 8'hE0 && ps2_key_data_3 == 8'h23)
            right <= 1'b1;
        else if (ps2_key_data_1 == 8'hE0 && ps2_key_data_2 == 8'hF0 && ps2_key_data_3 == 8'h23)
            right <= 1'b0;
    end
    PS2_Controller PS2 (CLOCK_50,reset,PS2_CLK,PS2_DAT,ps2_key_data,ps2_key_pressed);
endmodule
```