University of Nevada Las Vegas. Department of Electrical and Computer Engineering Laboratories.

Class:	**CPE200L 1001**	Semester:	**Spring 2024**
Points		Document author:	**Alireza Bolourian**
		Author's email:	**bolouria@unlv.nevada.edu**
		Document topic:	**Mid-Project Report**
Instructor's comments:			

Goal: The goal of this project is to become better familiarized with coding in SystemVerilog and the connections between the modules while making an interactive game utilizing the FPGA board and visual display using a VGA.

Abstract: A spaceship is stranded in space. Asteroids move toward the spaceship in increasing amount that spawn in randomized fashion. The spaceship has to dodge these asteroids as long as possible.

Project Overview:

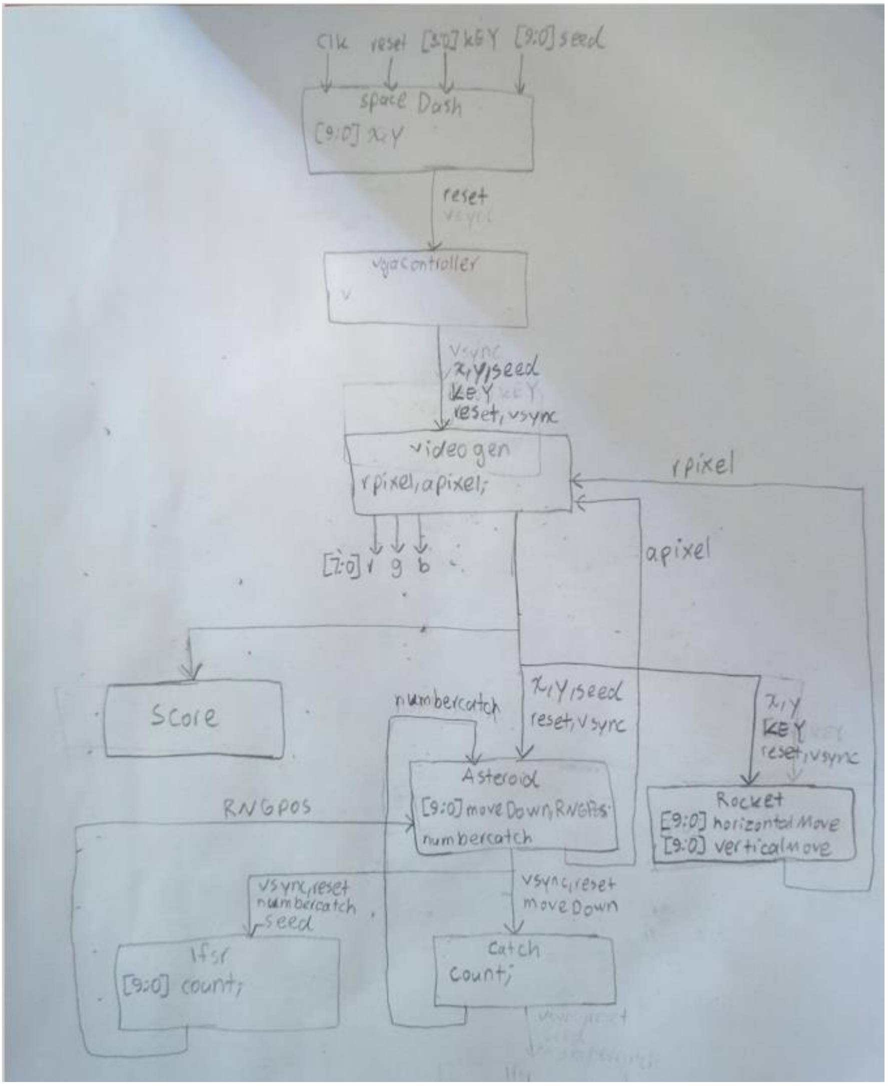
- **Top-Level Module:**
  - Manages all major components of the game.
  - Instantiates other modules and handles signal routing to other modules.
- **VGAController and VideoGen Module**
  - Generate timing signals for pixel display.
- **Rocket Module:**
  - Defines the rocket's visual representation.
  - Handles player input into movement of the rocket
- **Catch Module**
  - Generate numbercatch signal for the lfsr module.
- **LFSR module**
  - Generate a random number for the position of asteroid to spawn at
- **Asteroid Module:**
  - Randomly spawns asteroids at initial positions.

    - Controls the movement pattern (straight down)
- **Scoring/UI Module:**
    - Keeps track of the player's score.
    - Collision detection logic.
    - May handle additional UI elements like a "Game Over" screen.

Lessons Learned:

I have attempted to connect a joystick with buttons to control the rocket movements in four directions. However, I realize that the buttons are not one-to-one and mapping the buttons to movement is more complicated. As the buttons are not an essential component of the project, I will reattempt to connect the joystick near the end of project if time allows or potentially make a custom switch circuit on a breadboard.

Block Diagram



Appendix: SystemVerilog Code

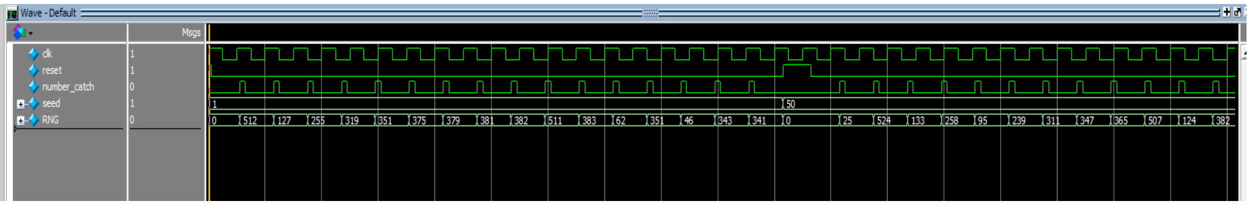
```
// vga.sv

module vga(input  logic clk, reset,
           input  logic keyright, keyleft, keyup, keydown,
           input  logic [9:0] seed,
                     output logic vcc,
           output logic vgaclk,          // 25.175 MHz VGA clock
           output logic hsync, vsync,
           output logic sync_b, blank_b, // to monitor & DAC
           output logic [7:0] r, g, b);  // to video DAC

  logic [9:0] x, y;

  // divide 50 MHz input clock by 2 to get 25 MHz clock
  always_ff @(posedge clk, posedge reset)
    if (reset)
         vgaclk = 1'b0;
    else
         vgaclk = ~vgaclk;

  // generate monitor timing signals
  vgaController vgaCont(vgaclk, reset, hsync, vsync, sync_b, blank_b, x, y);

  // user-defined module to determine pixel color
  videoGen videoGen(x, y, seed, keyright, keyleft, keyup, keydown, vsync, reset, r,
g, b);

endmodule


module vgaController #(parameter HBP     = 10'd48,   // horizontal back porch
                                 HACTIVE = 10'd640,  // number of pixels per line
                                 HFP     = 10'd16,   // horizontal front porch
                                 HSYN    = 10'd96,   // horizontal sync pulse = 60
to move electron gun back to left
                                 HMAX    = HBP + HACTIVE + HFP + HSYN,
//48+640+16+96=800: number of horizontal pixels (i.e., clock cycles)
                                 VBP     = 10'd32,   // vertical back porch
                                 VACTIVE = 10'd480,  // number of lines
                                 VFP     = 10'd11,   // vertical front porch
                                 VSYN    = 10'd2,    // vertical sync pulse = 2 to
move electron gun back to top
                                 VMAX    = VBP + VACTIVE + VFP  + VSYN)
//32+480+11+2=525: number of vertical pixels (i.e., clock cycles)

     (input  logic vgaclk, reset,
      output logic hsync, vsync, sync_b, blank_b,
      output logic [9:0] hcnt, vcnt);

      // counters for horizontal and vertical positions
      always @(posedge vgaclk, posedge reset) begin
        if (reset) begin
          hcnt <= 0;
          vcnt <= 0;
        end
        else  begin
```

```
          hcnt++;
         if (hcnt == HMAX) begin
            hcnt <= 0;
              vcnt++;
              if (vcnt == VMAX)
                vcnt <= 0;
          end
        end
      end


      // compute sync signals (active low)
      assign hsync  = ~( (hcnt >= (HACTIVE + HFP)) & (hcnt < (HACTIVE + HFP + HSYN))
);
      assign vsync  = ~( (vcnt >= (VACTIVE + VFP)) & (vcnt < (VACTIVE + VFP + VSYN))
);
      // assign sync_b = hsync & vsync;
      assign sync_b = 1'b0;  // this should be 0 for newer monitors

      // force outputs to black when not writing pixels
      // The following also works: assign blank_b = hsync & vsync;
      assign blank_b = (hcnt < HACTIVE) & (vcnt < VACTIVE);
endmodule


module videoGen(input logic [9:0] x, y, seed,
                                input logic keyright, keyleft, keyup, keydown,
vsync, reset,
                                output logic [7:0] r, g, b);
  logic rpixel, apixel;

  rocket r1(x, y, vsync, reset, keyright, keyleft, keyup, keydown, rpixel);
  asteroid a1(x, y, seed, reset, vsync, apixel);

  assign {r, g, b} = rpixel ? 24'hFFFFFF : apixel ? 24'h8968CD : 24'h000000;

endmodule

// display the rocket at the bottom middle of the screen
module rocket(input logic [9:0] x, y,
                                input logic vsync, reset, keyright, keyleft, keyup,
keydown,
                           output logic rpixel);

    // Data Structure for Rocket Shape
    logic [15:0][14:0] rocket_shape = {
       15'b100000010000001,
      15'b110000010000011,
       15'b111000111000111,
      15'b111110111011111,
       15'b101111111111101,
      15'b100111010111001,
       15'b000111000111000,
      15'b000111101111000,
      15'b000101111101000,
      15'b000000111000000,
       15'b000000111000000,
      15'b000000111000000,
      15'b000000111000000,
      15'b000000111000000,
      15'b000000010000000,
      15'b000000010000000,
      15'b000000010000000
    };


        logic [9:0] xleft, xright, ytop, ybottom, horizontalMove, verticalMove;

        // Horizontal Movement
  always_ff @(posedge vsync, posedge reset) begin
             if (reset)
                   horizontalMove <= 0;
             else if ((keyright) && (horizontalMove+10'd325 < 10'd633))
                   horizontalMove <= horizontalMove + 10'd1;
             else if ((keyleft) && (horizontalMove+10'd325 > 10'd18))
                   horizontalMove <= horizontalMove - 10'd1;
             else
                   horizontalMove <= horizontalMove;
      end

      // Vertical Movement
      always_ff @(posedge vsync, posedge reset) begin
             if (reset)
                   verticalMove <= 0;
             else if ((keyup) && (verticalMove+10'd460 <= 10'd460))
                   verticalMove <= verticalMove + 10'd1;
             else if ((keydown) && (verticalMove +10'd460 >= 10'd30))
                   verticalMove <= verticalMove - 10'd1;
             else
                   verticalMove <= verticalMove;
      end


        assign xleft = 10'd312;
        assign xright = 10'd326;
        assign ytop = 10'd452;
        assign ybottom = 10'd468;

  // Inside rocket module, assuming rocket starts at X=315, Y=460
    always_comb begin
        if ((x-horizontalMove >= xleft) && (x-horizontalMove <= xright) &&
            (y-verticalMove >= ytop) && (y-verticalMove < ybottom) &&
            (rocket_shape[y-ytop-verticalMove][x-xleft-horizontalMove]))  begin
                rpixel = 1; // White
            end
        else begin
            rpixel = 0; // Black
        end
    end
endmodule

module asteroid(input logic [9:0] x, y, seed,
                                       input logic reset, vsync,
                output logic apixel);

// Data Structure for Asteroid Shape
```

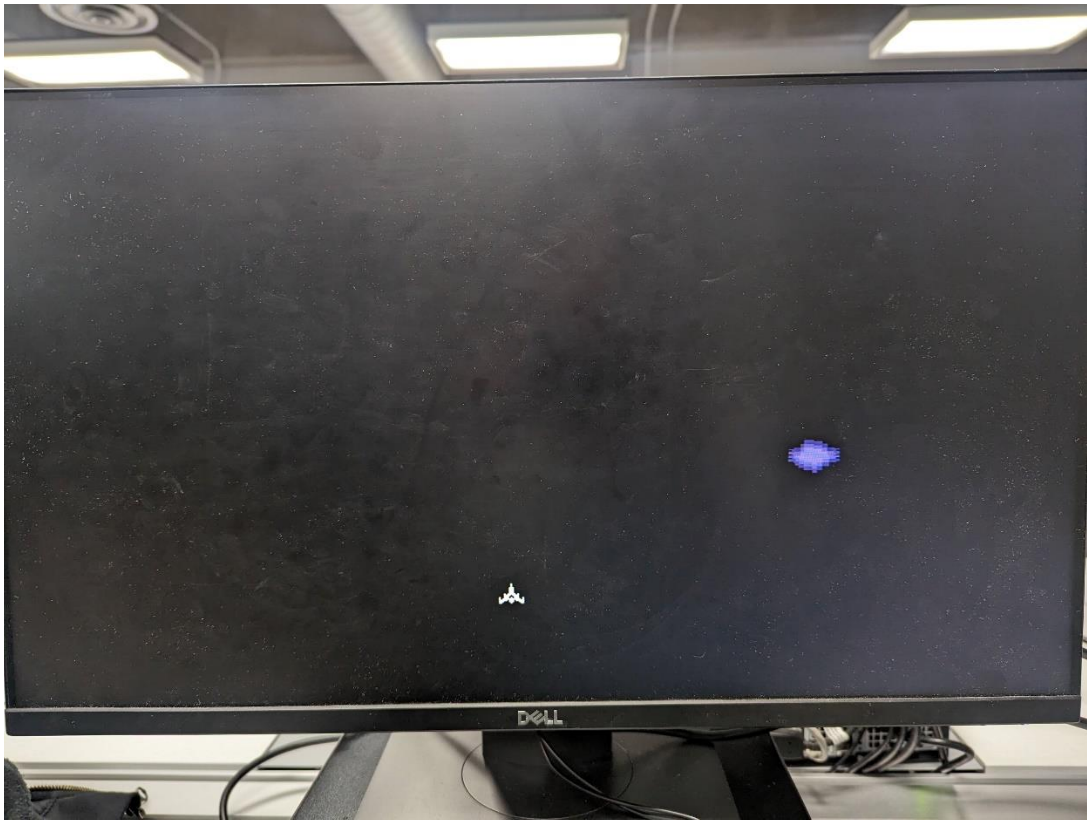
```
logic [19:0][29:0] asteroid_shape = {
      30'b000000000000011100000000000000,
      30'b000000000000011100000000000000,
      30'b000000000011111111111000000000,
      30'b000000000011111111111000000000,
      30'b000000000011111111111111000000,
      30'b000000111111111111111111000000,
      30'b000000111111111111111111110000,
      30'b000000111111111111111111110000,
      30'b001111111111111111111111111000,
      30'b111111111111111111111111111111,
       30'b111111111111111111111111111111,
      30'b001111111111111111111111111100,
      30'b111111111111111111111111111111,
      30'b000111111111111111111111111000,
      30'b000000011111111111111110000000,
      30'b000000011111111111111110000000,
       30'b000000011111111111111110000000,
       30'b000000000011111111111000000000,
       30'b000000000011111111111000000000,
       30'b000000000000000111111000000000
};

        logic [9:0] xleft, xright, ytop, ybottom, RNGpos, moveDown;
        logic numbercatch;

        catch c1(vsync, reset, moveDown, numbercatch);

     // Asteroid moving down
       always_ff @(posedge vsync, posedge reset) begin
       if (reset)
             moveDown <= 0;
    else if (moveDown == 10'd480)
        moveDown <= 0;
       else
             moveDown <= moveDown + 10'd1;
    end

        lfsr lfsr1(vsync, reset, numbercatch, seed, RNGpos);

        assign xleft = RNGpos;
        assign xright = RNGpos + 10'd30;
        assign ytop = 10'd0;
        assign ybottom = 10'd20;


  // Inside asteroid module, assuming asteroid starts at X=310, Y=0
  always_comb begin
       if ((x >= xleft) && (x < xright) &&
        (y-moveDown+10'd470 >= ytop+10'd470) && (y-moveDown+10'd470 <
ybottom+10'd470) &&
        (asteroid_shape[y-ytop-moveDown][x-xleft])) begin
         apixel = 1;
       end
       else begin
         apixel = 0;
       end
  end
```

```
endmodule

module lfsr (input logic clk, reset,
                          input logic number_catch,
                          input logic [9:0] seed,
                          output logic [9:0] RNG);

      logic [9:0] count;

      always_ff @(posedge clk, posedge reset) begin
             if (reset)                          count <= seed;
             else                                count <= {count[0] ^ count[9],
count[9:1]};
      end

      always_ff @(posedge number_catch, posedge reset) begin
             if (reset)                          RNG <= 10'd0;
             else                              RNG <= count % 10'd641;
      end

endmodule

module catch (input logic vsync, reset,
                                input logic [9:0] moveDown,
                                output logic numbercatch);
      logic count;

      always_ff @(posedge vsync, posedge reset) begin
             if (reset)
                          count <= 1'b1;
       else if (moveDown == 10'd479)
            count <= 1'b1;
        else
            count <= 1'b0;
    end

assign numbercatch = count;

endmodule
```

This systemVerilog code contains the rocket module which displays the rocket and updates the location of the rocket at about 60 frames per second based on the inputs from [3:0]KEY of the FPGA which corresponds to right, left, up and down movements. The logic also accounts for preventing the rocket from moving outside the screen. The systemverilog code includes the asteroid module which travels down on the screen and each time it travels all the way to the bottom, it respawns back at the top at a random pixel by catching a number from the lfsr module.



This testbench tests the lfsr module which shows that random numbers are selected for the pixel number to spawn the asteroid.



This image displays the rocket with the white color and the asteroid moving down the screen.