Final Project

ECE 5730

Andrew Gerber, Joseph Lamb

December 14, 2022

Overview

Design and build a simplified "Tetris"-style video game using the DE10-Lite FPGA board and a VGA monitor. The requirements for this project were: 1. The digital design must be done in VHDL. 2. The game image shall be 640 pixels wide by 480 pixels tall. 3. The two push buttons on the DE10-Lite board shall control game play. One button resets the game, the other starts the game. 4. Upon reset, a U-shaped enclosure with internal dimensions of 288 pixels wide by 448 pixels high shall be drawn with 1-pixel-wide white lines, centered roughly-horizontally on the screen. To the right of the enclosure 6 digits of score should appear in a white font, sized appropriately. 5. Areas of the screen not specified in requirement 4 shall be initially black. 6. When the "start" button is pressed a 32-pixel by 32-pixel cube will drop from the top of the screen. The color of the cube shall be randomly-generated from the set of {red, blue, green, yellow}. Pick an appropriate rate of movement for the cube to make the game challenging. 7. The user can move the cube from side-to-side inside the U-shaped enclosure as it descends. The cube movement can be controlled by either a potentiometer interfaced to the FPGA's ADC, or via the on-board accelerometer. As the enclosure is the exact width to hold 9 cubes, the horizontal position of any cube should always be in one of the 9 "lanes". Cubes must remain inside the enclosure at all times. 8. The cube continues to descend until it is resting on top of either (1) the bottom of the enclosure or (2) an already-stationary cube. If the cube comes to rest with it's top within 2 cube heights of the top of the screen, the game is over. When the game is over, no additional cubes should appear and the score for the game should remain visible until the reset button is pressed. 9. If at any point in time there is a horizontal or vertical sequence of at least 3 stationary cubes of the same color, those cubes will disappear and any higher cubes will fall down to fill in the gaps. 1 point should be added to the displayed score for each cube that disappears. If the rearranging causes more sequences of 3 or more, those cubes should also disappear. The object of the game is to maximize the score. 10. Once all remaining cubes are stationary, the next cube begins its descent from the top of the screen. 11. Produce 4 unique sounds using the FPGA and an external speaker/buzzer. One when a cube is moved a slot to the left or right, one when a cube becomes stationary, one when a set of 3 or more cubes disappears, and one when the game is over.

Procedure

The design was divided into three main submodules: GameEngine, GraphicsEngine, and VGA. The system block diagram is shown in Figure 1.

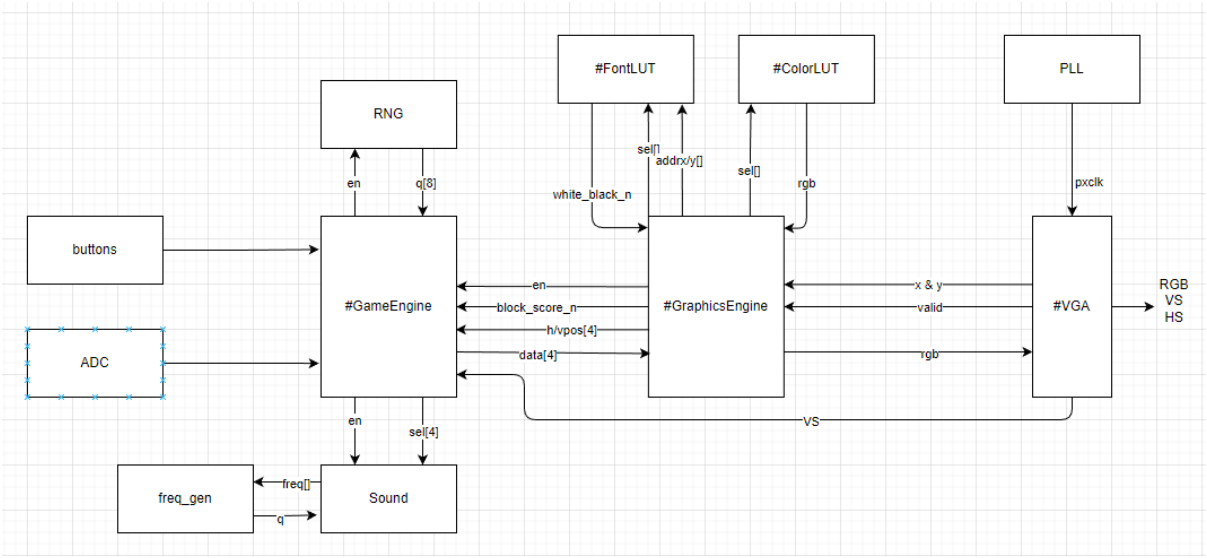


Figure 1. System block diagram

The GameEngine handled the movement of blocks within the enclosure, the position of static blocks, and the player's score. The enclosure was constantly scanned for any "floating" blocks, or blocks that were not stationary on another block or at the bottom of the enclosure. Once all blocks were found to be stationary, the enclosure was then scanned for any horizontal or vertical sequence of 3 cubes of the same color. Sequences of 3 cubes of the same color were cleared from the screen, the score was incremented, and the scan for floating blocks resumed. This search pattern continued until it was determined that there were no floating blocks or sequences of 3. After the scan, the game would check if there were any blocks within two spaces of the top of the enclosure to determine if the game was over. If the game was not over, a request was sent to the RNG for the color of the next falling generated. The block movement between the 9 lanes were determined from the input of the ADC. The 4 MSB were taken from the ADC input vector and values 0 through 8 were assigned to each respective lane. The block stayed in the right-most lane for input values greater than 8.

Difficulties with the GameEngine mostly consisted of data structures. A two dimensional array was used to track block positions within the enclosure. While scanning the array for blocks to descend and clear, much attention was needed when indexing these positions. Much testing was done using ModelSim to check whether blocks were descending correctly and were cleared when intended. Before moving a block, the next space needed to be checked as to whether it was already occupied by another block. When passing a block color from the GameEngine to the GraphicsEngine, the color was given a code value. To distinguish between the single, falling, controllable block, a '1' was appended to the code which was removed once it became stationary.

The GraphicsEngine has two state machines. A block diagram of this sub-system is shown in Figure 2. The first one fetches the block colors codes and score from the GameEngine. It then decodes the block colors codes to RGB values, and the decimal score to a font matrix. The block RGB and font are each inserted to their respective FIFOs. The second state machine removes these values from the FIFOs and draws them to the screen. The fetch state machine caused the most problems in the design, and never got the score to work properly. Import parts of the intended behavior of the two state machines are shown in Figures 3 and 4.

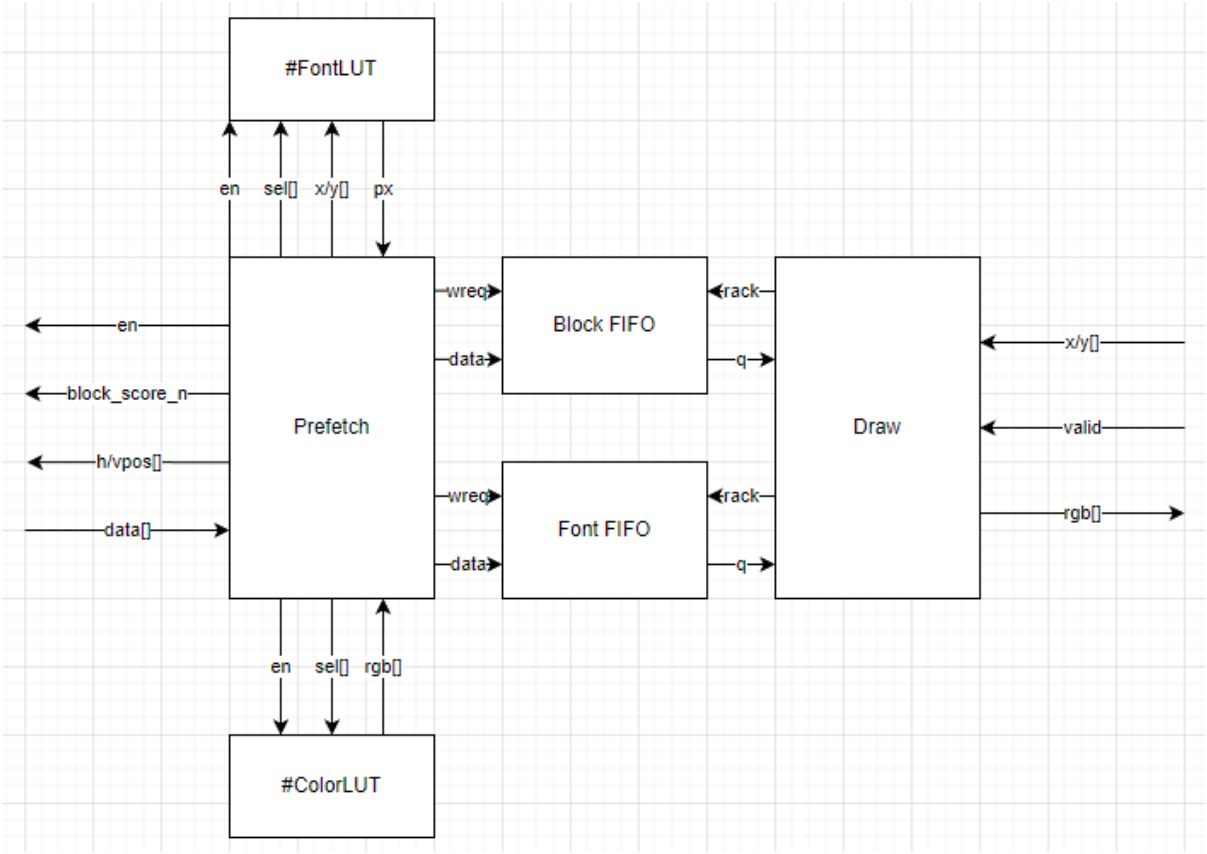


Figure 2. GraphicsEngine block diagram

Prefetch FSM Waveform

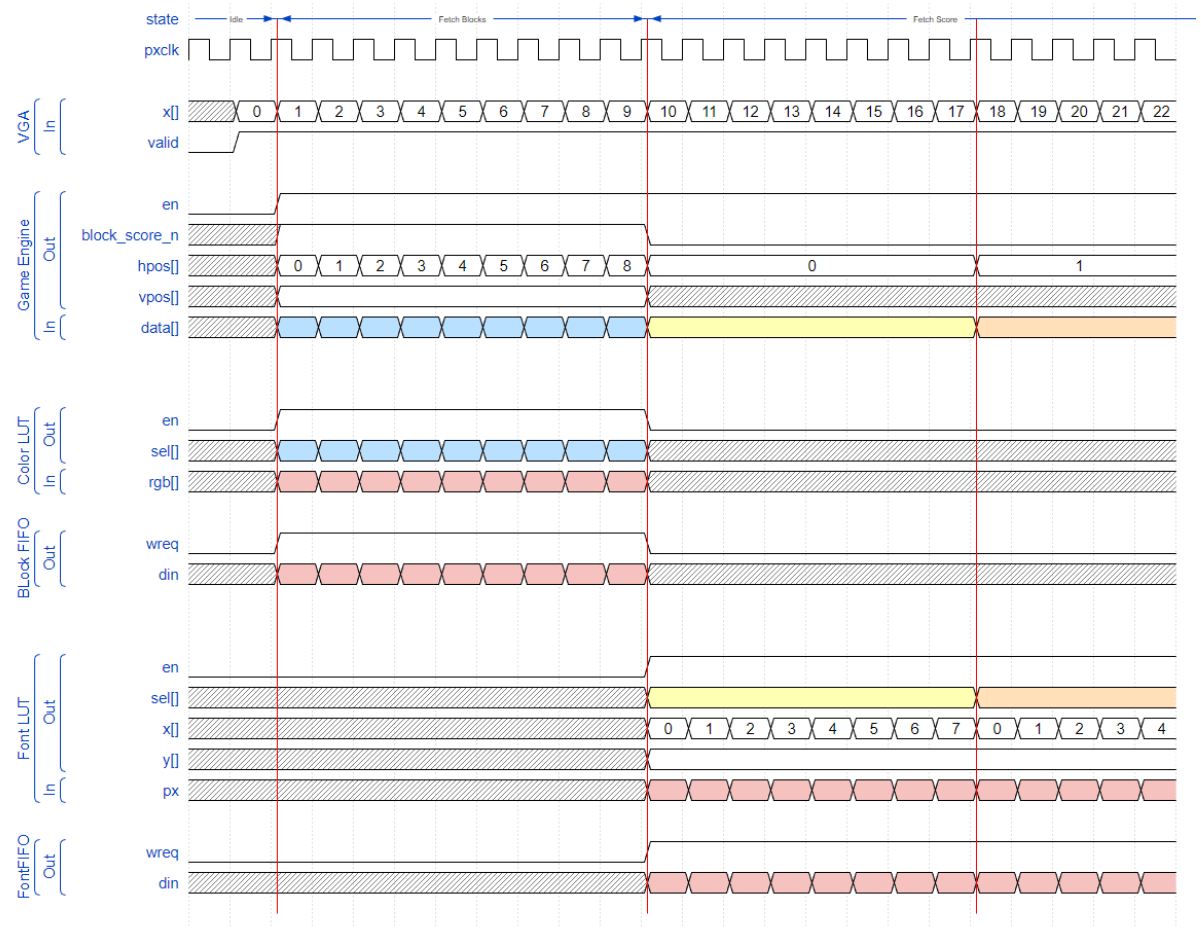
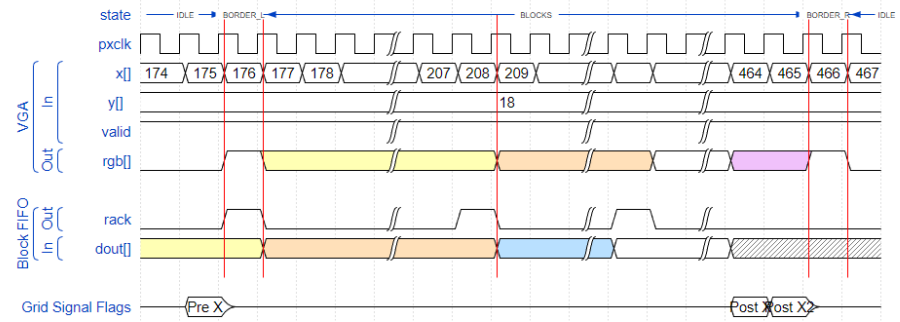


Figure 3. Prefetch waveform

Block Draw



Score Draw

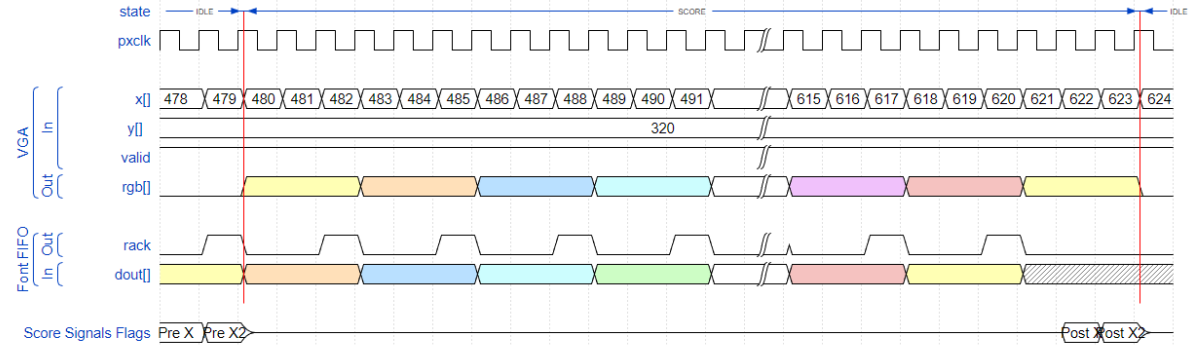


Figure 4. Draw waveform

Results

Initial tests on the monitor showed a stack of various colors on the far left edge of the enclosure. The rate of descent needed to be adjusted, but this showed that the game ended once the stack of blocks reached the top of the screen. After decreasing the rate of block descent, blocks were seen to be falling on the left edge of the screen. Once the ADC was implemented, the blocks could be moved horizontally as they fell. Three consecutive colored blocks were removed correctly from the screen.

The score would not display properly. A series of zeros would display, but when sending other numbers the score would print several numbers in one position. The sounds were not implemented as the majority of design time went into displaying the game and making the game functional. The ADC was trivial to implement as the design was brought from a previous lab. The RNG worked as expected, implementing a pseudo-random number generator.

Conclusion

It was essential to design the working of the system before beginning implementation. This included breaking the system into multiple sub-systems and defining their interfaces to allow them to be easily connected. One part of the initial design that was notably helpful was the Graphics engine wave forms in Figures 3 and 4.

Appendix A

src/top.vhd

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity Top is
    port (
        ADC_CLK_10    : in std_logic;
        MAX10_CLK1_50 : in std_logic;
        MAX10_CLK2_50 : in std_logic;

        HEX0 : out std_logic_vector(7 downto 0);
        HEX1 : out std_logic_vector(7 downto 0);
        HEX2 : out std_logic_vector(7 downto 0);
        HEX3 : out std_logic_vector(7 downto 0);
        HEX4 : out std_logic_vector(7 downto 0);
        HEX5 : out std_logic_vector(7 downto 0);

        KEY : in std_logic_vector(1 downto 0);

        LEDR : out std_logic_vector(9 downto 0);

        SW : in std_logic_vector(9 downto 0);

        VGA_R  : out std_logic_vector(3 downto 0);
        VGA_G  : out std_logic_vector(3 downto 0);
        VGA_B  : out std_logic_vector(3 downto 0);
        VGA_HS : out std_logic;
        VGA_VS : out std_logic;

        GSENSOR_CS_N : out std_logic;
        GSENSOR_INT  : in std_logic;
        GSENSOR_SCLK : out std_logic;
        GSENSOR_SDI  : in std_logic;
        GSENSOR_SDO  : out std_logic;

        ARDUINO_IO      : inout std_logic_vector(15 downto 0);
        ARDUINO_RESET_N : inout std_logic
    );
end entity Top;


architecture behavioral of Top is

    -------------------------------------------------
    -- BEGIN: COMPONENTS
    -------------------------------------------------
    component ADC_PLL
```

```
    port (
        inclk0  : in std_logic := '0';
        c0      : out std_logic;
        locked  : out std_logic
    );
end component;
signal adc_pll_clk    : std_logic;
signal adc_pll_locked : std_logic;

component PLL is
    port (
        areset  : in std_logic := '0';
        inclk0  : in std_logic := '0';
        c0      : out std_logic
    );
end component;
signal pxclk : std_logic;

component ADC_Controller is
    generic (
        CLK_FREQ : integer;
        SAMPLE_FREQ : integer
    );
    port (
        clk             : in std_logic;
        rst_n           : in std_logic;
        adc_pll_clk     : in std_logic;
        adc_pll_locked  : in std_logic;
        dout            : out std_logic_vector(11 downto 0)
    );
end component ADC_Controller;

component GameEngine is
    port (
        clk   : in std_logic;
        rst_n : in std_logic;
        VS    : in std_logic;

        -- Graphics Engine port group
        game_en          : in std_logic;
        game_blk_score_n : in std_logic;
        game_hpos        : in unsigned(3 downto 0);
        game_vpos        : in unsigned(3 downto 0);
        game_data        : out std_logic_vector(3 downto 0);

        -- Audio Engine port group
        aud_en  : out std_logic;
        aud_sel : out std_logic_vector(3 downto 0);

        -- Controls port group
        start_btn   : in std_logic;
```

```
        dir_control : in std_logic_vector(11 downto 0)
    );
end component GameEngine;
signal game_en          : std_logic;
signal game_blk_score_n : std_logic;
signal game_hpos        : unsigned(3 downto 0);
signal game_vpos        : unsigned(3 downto 0);
signal game_data        : std_logic_vector(3 downto 0);
signal start_btn        : std_logic;
signal dir_control      : std_logic_vector(11 downto 0);

component GraphicsEngine is
    port (
        pxclk : in std_logic;
        rst_n : in std_logic;

        -- Game Engine port group
        game_en          : out std_logic;
        game_blk_score_n : out std_logic;
        game_hpos        : out unsigned(3 downto 0);
        game_vpos        : out unsigned(3 downto 0);
        game_data        : in std_logic_vector(3 downto 0);

        -- VGA port group
        vga_x     : in unsigned(9 downto 0);
        vga_y     : in unsigned(8 downto 0);
        vga_valid : in std_logic;
        vga_rgb   : out std_logic_vector(23 downto 0)
    );
end component GraphicsEngine;
signal vga_rgb : std_logic_vector(23 downto 0);

component VGA is
    port (
        pxclk : in std_logic;
        rst_n : in std_logic;
        xaddr : out unsigned(9 downto 0);
        yaddr : out unsigned(8 downto 0);
        addr_valid : out std_logic;
        HS : out std_logic;
        VS : out std_logic
    );
end component VGA;
signal vga_x     : unsigned(9 downto 0);
signal vga_y     : unsigned(8 downto 0);
signal vga_valid : std_logic;
-------------------------------------------------
-- END: COMPONENTS
-------------------------------------------------
```

```
    -------------------------------------------------
    -- BEGIN: INTERNAL SIGNALS
    -------------------------------------------------
    signal rst   : std_logic;
    signal rst_n : std_logic;
    signal VS    : std_logic;
    -------------------------------------------------
    -- END: INTERNAL SIGNALS
    -------------------------------------------------

begin

    -------------------------------------------------
    -- BEGIN: WIRING
    -------------------------------------------------
    rst   <= not key(0);
    rst_n <= key(0);
    start_btn <= not key(1);
    VGA_R <= vga_rgb(23 downto 20) when (vga_valid = '1') else (others => '0');
    VGA_G <= vga_rgb(15 downto 12) when (vga_valid = '1') else (others => '0');
    VGA_B <= vga_rgb(7  downto 4)  when (vga_valid = '1') else (others => '0');
    VS    <= VGA_VS;
    -------------------------------------------------
    -- END: WIRING
    -------------------------------------------------

    -------------------------------------------------
    -- BEGIN: INSTANTIATIONS
    -------------------------------------------------
    PLL0 : ADC_PLL
        port map (
            inclk0  => ADC_CLK_10,
            c0      => adc_pll_clk,
            locked  => adc_pll_locked
        );

    PLL_1 : PLL
        port map (
            areset  => '0',
            inclk0  => MAX10_CLK1_50,
            c0      => pxclk
        );

    ADC_Ctrl_0 : ADC_Controller
        generic map (
            CLK_FREQ => 50_000_000,
            SAMPLE_FREQ => 10_000
        )
        port map(
            clk             => MAX10_CLK1_50,
            rst_n           => rst_n,
```

```
            adc_pll_clk     => adc_pll_clk,
            adc_pll_locked  => adc_pll_locked,
            dout            => dir_control
        );

    GameEngine_0 : GameEngine
        port map (
            clk             => pxclk,
            rst_n           => rst_n,
            VS              => VS,
            game_en         => game_en,
            game_blk_score_n=> game_blk_score_n,
            game_hpos       => game_hpos,
            game_vpos       => game_vpos,
            game_data       => game_data,
            start_btn       => start_btn,
            dir_control     => dir_control
        );

    GraphicsEngine_0 : GraphicsEngine
        port map (
            pxclk            => pxclk,
            rst_n            => rst_n,
            game_en          => game_en,
            game_blk_score_n => game_blk_score_n,
            game_hpos        => game_hpos,
            game_vpos        => game_vpos,
            game_data        => game_data,
            vga_x            => vga_x,
            vga_y            => vga_y,
            vga_valid        => vga_valid,
            vga_rgb          => vga_rgb
        );

    VGA_0 : VGA
        port map (
            pxclk       => pxclk,
            rst_n       => rst_n,
            xaddr       => vga_x,
            yaddr       => vga_y,
            addr_valid  => vga_valid,
            HS          => VGA_HS,
            VS          => VGA_VS
        );
    ------------------------------------------------
    -- END: INSTANTIATIONS
    ------------------------------------------------

end architecture behavioral;
```

Appendix B

src/GameEngine.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity GameEngine is
  port (
    clk   : in std_logic;
    rst_n : in std_logic;
    VS    : in std_logic;

    -- Graphics Engine port group
    game_en        : in std_logic;
    game_blk_score_n : in std_logic;
    game_hpos        : in unsigned(3 downto 0);
    game_vpos        : in unsigned(3 downto 0);
    game_data      : out std_logic_vector(3 downto 0);

    -- Audio Engine port group
    aud_en  : out std_logic;
    aud_sel : out std_logic_vector(3 downto 0);

    -- Controls port group
    start_btn   : in std_logic;
    dir_control : in std_logic_vector(11 downto 0)
  );
end entity GameEngine;

architecture behavioral of GameEngine is

  -- testing RNG
  component rng is
    port (
        clk : in std_logic;
        rst_n : in std_logic;
        rng_en : in std_logic;
        rng_q  : out std_logic_vector(7 downto 0)
    );
  end component;

  signal x, y : unsigned(3 downto 0);
  -- 0 black
  -- 1 red
  -- 2 green
  -- 3 blue
  -- 4 yellow
  type BLOCK_ROW_T is array(0 to 8) of std_logic_vector(3 downto 0);
  type BLOCK_ARR_T is array(0 to 13) of BLOCK_ROW_T;
  signal BLOCK_ARR : BLOCK_ARR_T := (others => (others => x"0"));
```

```
  -- := (
  --   (x"0",x"1",x"2",x"3",x"4",x"0",x"1",x"2",x"3"), -- first row
  --   (x"1",x"2",x"3",x"4",x"0",x"1",x"2",x"3",x"4"), -- second row
  --   (x"2",x"3",x"4",x"0",x"1",x"2",x"3",x"4",x"0"), -- third row
  --   (x"3",x"4",x"0",x"1",x"2",x"3",x"4",x"0",x"1"), -- fourth row
  --   (x"4",x"0",x"1",x"2",x"3",x"4",x"0",x"1",x"2"), -- fifth row
  --   (x"0",x"1",x"2",x"3",x"4",x"0",x"1",x"2",x"3"), -- 6th row
  --   (x"1",x"2",x"3",x"4",x"0",x"1",x"2",x"3",x"4"), -- 7th row
  --   (x"2",x"3",x"4",x"0",x"1",x"2",x"3",x"4",x"0"), -- 8th row
  --   (x"3",x"4",x"0",x"1",x"2",x"3",x"4",x"0",x"1"), -- 9th row
  --   (x"4",x"0",x"1",x"2",x"3",x"4",x"0",x"1",x"2"), -- 10th row
  --   (x"0",x"1",x"2",x"3",x"4",x"0",x"1",x"2",x"3"), -- 11th row
  --   (x"1",x"2",x"3",x"4",x"0",x"1",x"2",x"3",x"4"), -- 12th row
  --   (x"2",x"3",x"4",x"0",x"1",x"2",x"3",x"4",x"0"), -- 13th row
  --   (x"3",x"4",x"0",x"1",x"2",x"3",x"4",x"0",x"1") -- 14th row
  -- );

  signal game_over, cleared : std_logic;
  signal floating_blocks : std_logic;

  signal rng_en : std_logic;
  signal rng_color : std_logic_vector(7 downto 0);

  signal dir : std_logic_vector(11 downto 8);
  signal lane : natural;

  type SCORE_ARR is array(0 to 5) of std_logic_vector(3 downto 0);
  signal score_data : SCORE_ARR := (others => x"0");
  signal score : unsigned(23 downto 0) := (others => '0');
  signal add_points : unsigned(7 downto 0);

  signal VS_prev : std_logic;
  signal descend_counter : natural;
  constant DESCEND_RATE : integer := 16; -- for simulation
  -- constant DESCEND_RATE : integer := 25_000_000;

begin

  -- testing RNG
  color_generator : rng
    port map (
        clk     => clk,
        rst_n   => rst_n,
        rng_en  => rng_en,
        rng_q   => rng_color
    );

  x <= game_hpos;
  y <= game_vpos;
  dir <= dir_control(11 downto 8);
  -- rng_color <= rng_q;
```

```
game_proc : process(clk)
begin
  if rising_edge(clk) then
    if (rst_n = '0') then
      game_over <= '1';
      floating_blocks <= '0';
      cleared <= '0';
      BLOCK_ARR <= (others => (others => x"0"));
      rng_en <= '0';
      descend_counter <= 0;
      add_points <= (others => '0');
    else
      if (start_btn = '1') then
        game_over <= '0';
      end if;

      VS_prev <= VS;
      if (VS = '1' and VS_prev = '0') then
          if (descend_counter = DESCEND_RATE) then
            rng_en <= '0';
            descend_counter <= 0;
            if (game_over = '0') then
              floating_blocks <= '0';
              for j in 0 to 12 loop
                for i in 0 to 8 loop
                  -- main descending block
                  if (BLOCK_ARR(j)(i)(3) = '1') then
                    -- BLOCK_ARR(j)(lane) <= BLOCK_ARR(j)(i);
                    -- BLOCK_ARR(j+1)(i) <= x"0";
                    if (BLOCK_ARR(j+1)(lane) /= x"0") then -- main block landed on anothe
                      BLOCK_ARR(j)(lane) <= "0" & BLOCK_ARR(j)(i)(2 downto 0);
                    elsif (j >= 12) then  -- main block landed on bottom
                      BLOCK_ARR(j+1)(lane) <= "0" & BLOCK_ARR(j)(i)(2 downto 0);
                      BLOCK_ARR(j)(i) <= x"0";
                    else
                      BLOCK_ARR(j+1)(lane) <= BLOCK_ARR(j)(i);
                      BLOCK_ARR(j)(i) <= x"0";
                      floating_blocks <= '1';
                    end if;
                  -- end if;

                  -- search for other floating blocks and drop them
                  elsif (BLOCK_ARR(j)(i) /= x"0" AND BLOCK_ARR(j+1)(i) = x"0") then
                    BLOCK_ARR(j+1)(i) <= BLOCK_ARR(j)(i);
                    BLOCK_ARR(j)(i) <= x"0";
                    floating_blocks <= '1';
                  end if;
                end loop;
              end loop;
```

```
        if (floating_blocks = '0') then
          -- search for and clear horizontal rows of 3
          cleared <= '0';
          for j in 13 downto 0 loop
            for i in 0 to 6 loop
              if (BLOCK_ARR(j)(i) /= x"0") then
                if (BLOCK_ARR(j)(i) = BLOCK_ARR(j)(i+1) AND BLOCK_ARR(j)(i) = BLOCK
                  BLOCK_ARR(j)(i) <= x"0";
                  BLOCK_ARR(j)(i+1) <= x"0";
                  BLOCK_ARR(j)(i+2) <= x"0";
                  cleared <= '1';
                  add_points <= add_points + 3;
                end if;
              end if;
            end loop;
          end loop;

          -- search for and clear vertical rows of 3
          for i in 0 to 8 loop
            for j in 13 downto 2 loop
              if (BLOCK_ARR(j)(i) /= x"0") then
                if (BLOCK_ARR(j)(i) = BLOCK_ARR(j-1)(i) AND BLOCK_ARR(j)(i) = BLOCK
                  BLOCK_ARR(j)(i) <= x"0";
                  BLOCK_ARR(j-1)(i) <= x"0";
                  BLOCK_ARR(j-2)(i) <= x"0";
                  cleared <= '1';
                  add_points <= add_points + 3;
                end if;
              end if;
            end loop;
          end loop;
        end if;

        -- game over or request new block color from rng
        if (cleared = '0' and floating_blocks = '0') then

          if (BLOCK_ARR(1) /= (x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0")) then
            game_over <= '1';
          else
            rng_en <= '1';
            floating_blocks <= '1';
          end if;
        end if;
      end if; -- game_over
    else
      descend_counter <= descend_counter + 1;
    end if;
  end if;

-- request block color from rng until we get a valid number between 1 and 4
if (rng_en = '1') then
```

```
        if (rng_color(3 downto 0) > x"0" and rng_color(3 downto 0) < x"5") then
          BLOCK_ARR(0)(lane) <= '1' & rng_color(2 downto 0); -- BLOCK_ARR(0)(4) should ev
          floating_blocks <= '1';
          rng_en <= '0';
        end if;
      end if;

      if (add_points > x"0") then
        for i in 0 to 4 loop
          score_data(i) <= std_logic_vector(unsigned(score_data(i)) + 1);
          if score_data(i) >= x"9" then
            score_data(i) <= (others => '0');
            score_data(i+1) <= std_logic_vector(unsigned(score_data(i+1)) + 1);
          else
            exit;
          end if;
        end loop;
          add_points <= add_points - 1;
      end if;

      if (dir = x"0") then
        lane <= 0;
      elsif (dir = x"1") then
        lane <= 1;
      elsif (dir = x"2") then
        lane <= 2;
      elsif (dir = x"3") then
        lane <= 3;
      elsif (dir = x"4") then
        lane <= 4;
      elsif (dir = x"5") then
        lane <= 5;
      elsif (dir = x"6") then
        lane <= 6;
      elsif (dir = x"7") then
        lane <= 7;
      elsif (dir >= x"8") then
        lane <= 8;
      end if;

    end if; -- rst_n
  end if; -- rising_edge(clk)
end process;


gfx_proc : process(clk)
begin
  if (rst_n = '0') then
    game_data <= (others => '0');
  else
    if (game_en = '1') then
```

```
        if (game_blk_score_n = '1') then -- send block color data
          case game_hpos is
            when x"0" => game_data <= block_arr(to_integer(game_vpos))(0);
            when x"1" => game_data <= block_arr(to_integer(game_vpos))(1);
            when x"2" => game_data <= block_arr(to_integer(game_vpos))(2);
            when x"3" => game_data <= block_arr(to_integer(game_vpos))(3);
            when x"4" => game_data <= block_arr(to_integer(game_vpos))(4);
            when x"5" => game_data <= block_arr(to_integer(game_vpos))(5);
            when x"6" => game_data <= block_arr(to_integer(game_vpos))(6);
            when x"7" => game_data <= block_arr(to_integer(game_vpos))(7);
            when x"8" => game_data <= block_arr(to_integer(game_vpos))(8);
            when others => game_data <= (others => '0');
          end case;
          -- data <= block_arr(to_integer(game_vpos))(to_integer(game_hpos)); -- Doesn't wo
        else
          game_data <= std_logic_vector(game_hpos);
        end if;
      else
        game_data <= (others => '0');
      end if;
    end if; -- (rst_n = '0')
  end process;

end architecture behavioral;
```

Appendix C

src/GraphicsEngine.vhd

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity GraphicsEngine is
    port (
        pxclk : in std_logic;
        rst_n : in std_logic;

        -- Game Engine port group
        game_en          : out std_logic;
        game_blk_score_n : out std_logic;
        game_hpos        : out unsigned(3 downto 0);
        game_vpos        : out unsigned(3 downto 0);
        game_data        : in std_logic_vector(3 downto 0);

        -- VGA port group
        vga_x     : in unsigned(9 downto 0);
        vga_y     : in unsigned(8 downto 0);
        vga_valid : in std_logic;
        vga_rgb   : out std_logic_vector(23 downto 0)
    );
end entity GraphicsEngine;


architecture behavioral of GraphicsEngine is

    ------------------------------------------------
    -- BEGIN: COMPONENTS
    ------------------------------------------------
    component ColorLUT is
        port (
            en  : in std_logic;
            sel : in std_logic_vector(2 downto 0);
            rgb : out std_logic_vector(23 downto 0)
        );
    end component ColorLUT;
    signal color_en  : std_logic;
    signal color_sel : std_logic_vector(2 downto 0);
    signal color_rgb : std_logic_vector(23 downto 0);

    component BlockFIFO is
        port (
            clock   : in std_logic;
            sclr    : in std_logic;
            wrreq   : in std_logic;
            data    : in std_logic_vector (23 downto 0);
```

```
        rdreq   : in std_logic;
        q       : out std_logic_vector(23 downto 0);
        full    : out std_logic;
        empty   : out std_logic
    );
end component BlockFIFO;
signal bfifo_wreq : std_logic;
signal bfifo_din  : std_logic_vector(23 downto 0);
signal bfifo_rack : std_logic;
signal bfifo_dout : std_logic_vector(23 downto 0);
signal bfifo_full : std_logic;
signal bfifo_empty: std_logic;


component FontLUT is
    port (
        en      : in std_logic;
        sel     : in unsigned(3 downto 0);
        xaddr   : in unsigned(2 downto 0);
        yaddr   : in unsigned(3 downto 0);
        px      : out std_logic
    );
end component FontLUT;
signal font_en  : std_logic;
signal font_sel : unsigned(3 downto 0);
signal font_x   : unsigned(2 downto 0);
signal font_y   : unsigned(3 downto 0);
signal font_px  : std_logic;

component FontFIFO is
    port (
        clock   : in std_logic;
        sclr    : in std_logic;
        wrreq   : in std_logic;
        data    : in std_logic_vector(0 downto 0);
        rdreq   : in std_logic;
        q       : out std_logic_vector(0 downto 0)
    );
end component FontFIFO;
signal ffifo_wreq : std_logic;
signal ffifo_din  : std_logic_vector(0 downto 0);
signal ffifo_rack : std_logic;
signal ffifo_dout : std_logic_vector(0 downto 0);
-------------------------------------------------
-- END: COMPONENTS
-------------------------------------------------


-------------------------------------------------
-- BEGIN: CONSTANTS
-------------------------------------------------
```

```
constant BLOCK_W    : natural := 32;
constant BLOCK_H    : natural := 32;
constant GRID_W     : natural := 9;
constant GRID_H     : natural := 14;
constant GRID_X     : natural := 177;
constant GRID_Y     : natural := 17;
constant GRID_X2    : natural := GRID_X + (BLOCK_W * GRID_W) - 1;
constant GRID_Y2    : natural := GRID_Y + (BLOCK_H * GRID_H) - 1;

constant BORDER_X   : natural := GRID_X - 1;
constant BORDER_Y   : natural := GRID_Y - 1;
constant BORDER_X2  : natural := GRID_X2 + 1;
constant BORDER_Y2  : natural := GRID_Y2 + 1;

constant FONT_W     : natural := 8;
constant FONT_H     : natural := 16;
constant FONT_SCALE : natural := 3;
constant SCORE_W    : natural := 6;
constant SCORE_X    : natural := 480;
constant SCORE_Y    : natural := 320;
constant SCORE_X2   : natural := SCORE_X + (FONT_SCALE * FONT_W * SCORE_W) - 1;
constant SCORE_Y2   : natural := SCORE_Y + (FONT_SCALE * FONT_H) - 1;
-------------------------------------------------
-- END: CONSTANTS
-------------------------------------------------


-------------------------------------------------
-- BEGIN: INTERNAL SIGNALS
-------------------------------------------------
signal rst             : std_logic;
signal is_blk_y        : std_logic;
signal is_score_y      : std_logic;
signal is_border_b_y   : std_logic;
signal is_pre_grid_x   : std_logic;
signal is_post_grid_x  : std_logic;
signal is_post_grid_x2 : std_logic;
signal is_pre_score_x  : std_logic;
signal is_pre_score_x2 : std_logic;
signal is_post_score_x : std_logic;
signal is_post_score_x2: std_logic;
-------------------------------------------------
-- END: INTERNAL SIGNALS
-------------------------------------------------


-------------------------------------------------
-- BEGIN: STATE MACHINES
-------------------------------------------------
-- Prefetch FSM
type PREFETCH_STATE_T is (
```

```
        PREFETCH_IDLE,
        FETCH_BLOCKS,
        FETCH_SCORE
    );
    signal prefetch_state       : PREFETCH_STATE_T;
    signal prefetch_hpos        : natural range 0 to GRID_W-1;
    signal prefetch_vpos        : natural range 0 to GRID_H-1;
    signal prefetch_blk_y       : natural range 0 to BLOCK_H-1;
    signal prefetch_font_y      : natural range 0 to FONT_H-1;
    signal prefetch_font_vscale : natural range 0 to FONT_SCALE-1;

    -- Draw FSM
    type DRAW_STATE_T is (
        DRAW_IDLE,
        DRAW_BORDER_L,
        DRAW_BLOCKS,
        DRAW_BORDER_R,
        DRAW_SCORE,
        DRAW_BORDER_B
    );
    signal draw_state  : DRAW_STATE_T;
    signal draw_blk_x  : natural range 0 to BLOCK_W-1;
    signal draw_hscale : natural range 0 to FONT_SCALE-1;
    ------------------------------------------------
    -- END: STATE MACHINES
    ------------------------------------------------

begin

    ------------------------------------------------
    -- BEGIN: INSTANTIATIONS
    ------------------------------------------------
    ColorLUT_0 : ColorLUT
        port map (
            en  => color_en,
            sel => color_sel,
            rgb => color_rgb
        );

    BlockFIFO_0 : BlockFIFO
        port map (
            clock   => pxclk,
            sclr    => rst,
            wrreq   => bfifo_wreq,
            data    => bfifo_din,
            rdreq   => bfifo_rack,
            q       => bfifo_dout,
            full    => bfifo_full,
            empty   => bfifo_empty
        );
```

```
FontLUT_0 : FontLUT
    port map (
        en    => font_en,
        sel   => font_sel,
        xaddr => font_x,
        yaddr => font_y,
        px    => font_px
    );

FontFIFO_0 : FontFIFO
    port map (
        clock   => pxclk,
        sclr    => rst,
        wrreq   => ffifo_wreq,
        data    => ffifo_din,
        rdreq   => ffifo_rack,
        q       => ffifo_dout
    );
-------------------------------------------------
-- END: INSTANTIATIONS
-------------------------------------------------



-------------------------------------------------
-- BEGIN: WIRING
-------------------------------------------------
rst             <= not rst_n;
is_blk_y        <= '1' when (vga_valid = '1') and (to_integer(vga_y) >= GRID_Y) and (to_
is_score_y      <= '1' when (vga_valid = '1') and (to_integer(vga_y) >= SCORE_Y) and (to
is_border_b_y   <= '1' when (vga_valid = '1') and (to_integer(vga_y) = BORDER_Y2) else
is_pre_grid_x   <= '1' when (vga_valid = '1') and (to_integer(vga_x) = GRID_X - 2) else
is_post_grid_x  <= '1' when (vga_valid = '1') and (to_integer(vga_x) = GRID_X2 - 1) else
is_post_grid_x2 <= '1' when (vga_valid = '1') and (to_integer(vga_x) = GRID_X2) else '0
is_pre_score_x  <= '1' when (vga_valid = '1') and (to_integer(vga_x) = SCORE_X - 2) else
is_pre_score_x2 <= '1' when (vga_valid = '1') and (to_integer(vga_x) = SCORE_X - 1) else
is_post_score_x <= '1' when (vga_valid = '1') and (to_integer(vga_x) = SCORE_X2 - 2) els
is_post_score_x2<= '1' when (vga_valid = '1') and (to_integer(vga_x) = SCORE_X2 - 1) els
-- END: WIRING
-------------------------------------------------

-------------------------------------------------
-- BEGIN: PREFETCH FSM
-------------------------------------------------
-- process (all)
-- begin
--     case prefetch_hpos is
--         when 2 => font_sel <= x"0";
--         when others => font_sel <= x"9";
--     end case;
-- end process;
game_hpos    <= to_unsigned(prefetch_hpos, game_hpos'length);
```

```
game_vpos    <= to_unsigned(prefetch_vpos, game_vpos'length);
color_sel    <= game_data(2 downto 0);
--font_sel     <= unsigned(game_data);
font_sel <= x"0";
bfifo_din    <= color_rgb;
ffifo_din(0) <= font_px;
prefetch_proc : process (pxclk)
begin
    if rising_edge(pxclk) then
        if (rst_n = '0') then
            prefetch_state  <= PREFETCH_IDLE;
            game_en         <= '0';
            color_en        <= '0';
            font_en         <= '0';
            bfifo_wreq      <= '0';
            ffifo_wreq      <= '0';
        else
            case prefetch_state is

                when PREFETCH_IDLE =>
                    if (vga_valid = '1') and (vga_x = 0) and (is_blk_y = '1') then
                        prefetch_state   <= FETCH_BLOCKS;
                        game_en          <= '1';
                        game_blk_score_n <= '1';
                        prefetch_hpos    <= 0;
                        color_en         <= '1';
                        bfifo_wreq       <= '1';
                        if (vga_y = GRID_Y) then
                            prefetch_vpos   <= 0;
                            prefetch_blk_y  <= 0;
                        else
                            if (prefetch_blk_y < BLOCK_H-1) then
                                prefetch_blk_y <= prefetch_blk_y + 1;
                            else
                                prefetch_vpos   <= prefetch_vpos + 1;
                                prefetch_blk_y  <= 0;
                            end if;
                        end if;
                    end if;

                when FETCH_BLOCKS =>
                    if (prefetch_hpos < GRID_W-1) then
                        prefetch_hpos <= prefetch_hpos + 1;
                    else
                        color_en   <= '0';
                        bfifo_wreq <= '0';
                        if (is_score_y = '1') then
                            prefetch_state   <= FETCH_SCORE;
                            game_blk_score_n <= '0';
                            prefetch_hpos    <= 0;
                            font_en          <= '1';
```

```
                            font_x          <= (others => '0');
                            ffifo_wreq      <= '1';
                            if (vga_y = SCORE_X) then
                                font_y              <= (others => '0');
                                prefetch_font_vscale <= 0;
                            else
                                if (prefetch_font_vscale < FONT_SCALE-1) then
                                    prefetch_font_vscale <= prefetch_font_vscale + 1;
                                else
                                    font_y              <= font_y + 1;
                                    prefetch_font_vscale <= 0;
                                end if;
                            end if;
                        else
                            prefetch_state <= PREFETCH_IDLE;
                            game_en        <= '0';
                        end if;
                    end if;

                when FETCH_SCORE =>
                    if (font_x < FONT_W-1) then
                        font_x    <= font_x + 1;
                    else
                        if (prefetch_hpos < SCORE_W-1) then
                            prefetch_hpos <= prefetch_hpos + 1;
                            font_x        <= (others => '0');
                        else
                            prefetch_state  <= PREFETCH_IDLE;
                            game_en         <= '0';
                            font_en         <= '0';
                            ffifo_wreq      <= '0';
                        end if;
                    end if;

                when others =>
                    prefetch_state  <= PREFETCH_IDLE;
                    game_en         <= '0';
                    color_en        <= '0';
                    font_en         <= '0';
                    bfifo_wreq      <= '0';
                    ffifo_wreq      <= '0';

            end case; -- prefetch_state
        end if; -- (rst_n = '0')
    end if; -- rising_edge(pxclk)
end process; -- prefetch_proc
-------------------------------------------------
-- END: PREFETCH FSM
-------------------------------------------------
```

```
------------------------------------------------
-- BEGIN: DRAW FSM
------------------------------------------------
draw_proc : process (pxclk)
begin
    if rising_edge(pxclk) then
        if (rst_n = '0') then
            draw_state  <= DRAW_IDLE;
            vga_rgb     <= x"000000";
            bfifo_rack  <= '0';
            ffifo_rack  <= '0';
        else
            case draw_state is

                when DRAW_IDLE =>
                    if (is_blk_y = '1') then
                        -- block row
                        if (is_pre_grid_x = '1') then
                            -- draw left border and get first block color
                            draw_state  <= DRAW_BORDER_L;
                            vga_rgb     <= x"FFFFFF";
                            bfifo_rack  <= '1';
                            draw_hscale <= 0;
                        end if;
                    end if;
                    if (is_score_y = '1') then
                        -- score row
                        if (is_pre_score_x = '1') then
                            -- get first score pixel
                            ffifo_rack  <= '1';
                        end if;
                        if (is_pre_score_x2 = '1') then
                            -- start drawing score
                            draw_state  <= DRAW_SCORE;
                            ffifo_rack  <= '0';
                            if (ffifo_dout = "1") then
                                vga_rgb <= x"FFFFFF";
                            else
                                vga_rgb <= x"000000";
                            end if;
                        end if;
                    end if;
                    if (is_border_b_y = '1') then
                        -- bottom border row
                        if (is_pre_grid_x = '1') then
                            -- start botton border
                            draw_state    <= DRAW_BORDER_B;
                            vga_rgb       <= x"FFFFFF";
                        end if;
                    end if;
```

```
when DRAW_BORDER_L =>
    draw_state  <= DRAW_BLOCKS;
    vga_rgb     <= bfifo_dout;
    bfifo_rack  <= '0';
    draw_blk_x  <= 0;

when DRAW_BLOCKS =>
    if (draw_blk_x < BLOCK_W-1) then
        draw_blk_x <= draw_blk_x + 1;
    else
        -- end of block
        if (is_post_grid_x2 = '1') then
            -- end of grid
            draw_state  <= DRAW_BORDER_R;
            vga_rgb     <= x"FFFFFF";
        else
            -- start next block
            vga_rgb     <= bfifo_dout;
            bfifo_rack  <= '0';
            draw_blk_x  <= 0;
        end if;
    end if;
    if (draw_blk_x = BLOCK_W-2) and (is_post_grid_x = '0') then
        -- get next block color if not end of grid
        bfifo_rack  <= '1';
    end if;

when DRAW_BORDER_R =>
    draw_state  <= DRAW_IDLE;
    vga_rgb     <= x"000000";

when DRAW_SCORE =>
    if (is_post_score_x2 = '1') then
        draw_state  <= DRAW_IDLE;
        vga_rgb     <= x"000000";
    else
        if (draw_hscale < FONT_SCALE-1) then
            draw_hscale <= draw_hscale + 1;
        else
            if (ffifo_dout = "1") then
                vga_rgb <= x"FFFFFF";
            else
                vga_rgb <= x"000000";
            end if;
            ffifo_rack  <= '0';
            draw_hscale <= 0;
        end if;
    end if;
    if (draw_hscale = FONT_SCALE-2) and (is_post_score_x = '0') then
        ffifo_rack  <= '1';
    end if;
```

```
                    when DRAW_BORDER_B =>
                        if (is_post_grid_x = '1') then
                            draw_state <= DRAW_BORDER_R;
                        end if;

                    WHEN others =>
                        draw_state  <= DRAW_IDLE;
                        vga_rgb     <= x"000000";
                        bfifo_rack  <= '0';
                        ffifo_rack  <= '0';

                end case; -- draw_state
            end if; -- (rst_n = '0')
        end if; -- rising_edge(pxclk)
    end process; -- draw_proc
    ------------------------------------------------
    -- END: DRAW FSM
    ------------------------------------------------

end architecture behavioral;
```