```
-- penalty_top.vhd
-- Nexys A7 penalty kick game
-- Uses external vga_sync.vhd (800x600 timing)
-- 60-second time limit, VGA graphics, 7-seg score / high score
-- Centered goal, random goalie dives, GOAL!/MISS! text
-- Power meter bar and gentler aiming

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

entity penalty_top is
    port(
        clk100   : in  std_logic;                 -- 100 MHz clock
        reset_n  : in  std_logic;                 -- active-low reset

        btnL     : in  std_logic;                 -- BTNL
        btnR     : in  std_logic;                 -- BTNR
        btnU     : in  std_logic;                 -- BTNU
        btnD     : in  std_logic;                 -- BTND
        btnC     : in  std_logic;                 -- BTNC (shoot)

        -- VGA
        vga_hs   : out std_logic;
        vga_vs   : out std_logic;
        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);

        -- 7-segment display (active-low on Nexys A7)
        an       : out std_logic_vector(7 downto 0);
        seg      : out std_logic_vector(6 downto 0)  -- g f e d c b a (active-low)
    );
end entity;

architecture rtl of penalty_top is

    ---------------------------------------------------------------------
    -- Comic text constants (used by font/text functions + renderer)
    ---------------------------------------------------------------------
    constant TXT_SCALE   : integer := 10;  -- bigger = more comic
    constant TXT_CHAR_W  : integer := 5 * TXT_SCALE;
    constant TXT_CHAR_H  : integer := 7 * TXT_SCALE;
    constant TXT_GAP     : integer := 2 * TXT_SCALE;
```

```
constant TXT_MSG_LEN : integer := 5;   -- "GOAL!" or "MISS!"

-----------------------------------------------------------------
-- Simple 5x7 font for needed characters (returns 5 bits per row)
-- Bit(4) is leftmost pixel of the 5-wide glyph.
-----------------------------------------------------------------
function font5x7(ch : character; row : integer) return std_logic_vector is
   variable bits : std_logic_vector(4 downto 0) := (others => '0');
begin
   case ch is
      when 'G' =>
         case row is
            when 0 => bits := "01110";
            when 1 => bits := "10001";
            when 2 => bits := "10000";
            when 3 => bits := "10111";
            when 4 => bits := "10001";
            when 5 => bits := "10001";
            when 6 => bits := "01110";
            when others => bits := "00000";
         end case;

      when 'O' =>
         case row is
            when 0 => bits := "01110";
            when 1 => bits := "10001";
            when 2 => bits := "10001";
            when 3 => bits := "10001";
            when 4 => bits := "10001";
            when 5 => bits := "10001";
            when 6 => bits := "01110";
            when others => bits := "00000";
         end case;

      when 'A' =>
         case row is
            when 0 => bits := "00100";
            when 1 => bits := "01010";
            when 2 => bits := "10001";
            when 3 => bits := "11111";
            when 4 => bits := "10001";
            when 5 => bits := "10001";
            when 6 => bits := "10001";
            when others => bits := "00000";
```

```
    end case;

when 'L' =>
    case row is
        when 0 => bits := "10000";
        when 1 => bits := "10000";
        when 2 => bits := "10000";
        when 3 => bits := "10000";
        when 4 => bits := "10000";
        when 5 => bits := "10000";
        when 6 => bits := "11111";
        when others => bits := "00000";
    end case;

when 'M' =>
    case row is
        when 0 => bits := "10001";
        when 1 => bits := "11011";
        when 2 => bits := "10101";
        when 3 => bits := "10101";
        when 4 => bits := "10001";
        when 5 => bits := "10001";
        when 6 => bits := "10001";
        when others => bits := "00000";
    end case;

when 'I' =>
    case row is
        when 0 => bits := "11111";
        when 1 => bits := "00100";
        when 2 => bits := "00100";
        when 3 => bits := "00100";
        when 4 => bits := "00100";
        when 5 => bits := "00100";
        when 6 => bits := "11111";
        when others => bits := "00000";
    end case;

when 'S' =>
    case row is
        when 0 => bits := "01111";
        when 1 => bits := "10000";
        when 2 => bits := "10000";
        when 3 => bits := "01110";
```

```
                when 4 => bits := "00001";
                when 5 => bits := "00001";
                when 6 => bits := "11110";
                when others => bits := "00000";
            end case;

        when '!' =>
            case row is
                when 0 => bits := "00100";
                when 1 => bits := "00100";
                when 2 => bits := "00100";
                when 3 => bits := "00100";
                when 4 => bits := "00100";
                when 5 => bits := "00000";
                when 6 => bits := "00100";
                when others => bits := "00000";
            end case;

        when ' ' =>
            bits := "00000";

        when others =>
            bits := "00000";
    end case;
    return bits;
end function;

--------------------------------------------------------------------
-- Message character selection ("GOAL!" or "MISS!")
--------------------------------------------------------------------
function msg_char(goal : std_logic; idx : integer) return character is
begin
    if goal = '1' then
        -- "GOAL!"
        case idx is
            when 0 => return 'G';
            when 1 => return 'O';
            when 2 => return 'A';
            when 3 => return 'L';
            when 4 => return '!';
            when others => return ' ';
        end case;
    else
        -- "MISS!"
```

```
        case idx is
            when 0 => return 'M';
            when 1 => return 'I';
            when 2 => return 'S';
            when 3 => return 'S';
            when 4 => return '!';
            when others => return ' ';
        end case;
    end if;
end function;

-----------------------------------------------------------------------
-- True if pixel (px,py) hits the filled text body (no outline).
-----------------------------------------------------------------------
function text_fill_pixel(goal : std_logic; px, py : integer) return boolean is
    variable cidx     : integer;
    variable local_x  : integer;
    variable cc, rr   : integer;
    variable glyph_ch : character;
    variable glyph_row: std_logic_vector(4 downto 0);

    constant CHAR_W   : integer := TXT_CHAR_W;
    constant CHAR_H   : integer := TXT_CHAR_H;
    constant SCALE    : integer := TXT_SCALE;
    constant GAP      : integer := TXT_GAP;
    constant MSG_LEN  : integer := TXT_MSG_LEN;
begin
    if (px < 0) or (py < 0) then
        return false;
    end if;

    cidx := px / (CHAR_W + GAP);
    if (cidx < 0) or (cidx >= MSG_LEN) then
        return false;
    end if;

    if py >= CHAR_H then
        return false;
    end if;

    local_x := px mod (CHAR_W + GAP);
    if local_x >= CHAR_W then
        return false;
    end if;
```

```
    cc := local_x / SCALE;  -- 0..4
    rr := py / SCALE;       -- 0..6

    glyph_ch  := msg_char(goal, cidx);
    glyph_row := font5x7(glyph_ch, rr);

    return (glyph_row(4-cc) = '1');
end function;

-----------------------------------------------------------------
-- True if pixel (px,py) hits the outline (halo around fill)
-----------------------------------------------------------------
function text_outline_pixel(goal : std_logic; px, py : integer) return boolean is
    variable nx, ny : integer;
    variable lx, ly : integer;
begin
    for ny in -1 to 1 loop
        for nx in -1 to 1 loop
            lx := px + nx;
            ly := py + ny;
            if text_fill_pixel(goal, lx, ly) then
                return true;
            end if;
        end loop;
    end loop;
    return false;
end function;

-----------------------------------------------------------------
-- Clock 100 MHz -> 25 MHz for VGA and game logic
-----------------------------------------------------------------
signal clk_div : unsigned(1 downto 0) := (others => '0');
signal clk25   : std_logic;

-----------------------------------------------------------------
-- VGA timing coming from external vga_sync
-----------------------------------------------------------------
signal hs, vs       : std_logic;
signal pixel_row_sv : std_logic_vector(10 downto 0);
signal pixel_col_sv : std_logic_vector(10 downto 0);
signal pixel_x      : unsigned(10 downto 0);
signal pixel_y      : unsigned(10 downto 0);
signal video_on     : std_logic;
```

```
---------------------------------------------------------------------
-- Frame tick (1 pulse per frame, from vsync)
---------------------------------------------------------------------
signal vs_d       : std_logic := '0';
signal frame_tick : std_logic := '0';

---------------------------------------------------------------------
-- 1-second tick (using clk25)
---------------------------------------------------------------------
signal sec_cnt      : unsigned(24 downto 0) := (others => '0');
signal one_sec_tick : std_logic := '0';
constant SEC_MAX    : unsigned(24 downto 0) := to_unsigned(24999999, 25);

---------------------------------------------------------------------
-- Button synchronisers (to clk25)
---------------------------------------------------------------------
signal btnL_sync, btnR_sync, btnU_sync, btnD_sync, btnC_sync : std_logic := '0';

---------------------------------------------------------------------
-- Game state + parameters
---------------------------------------------------------------------
type game_state_t is (AIM, SHOT, RESULT, GAMEOVER);
signal game_state : game_state_t := AIM;

constant SCREEN_W       : integer := 800;
constant SCREEN_H       : integer := 600;

constant BALL_R         : integer := 6;
constant GOAL_LINE_Y    : integer := 80;
constant PEN_SPOT_Y     : integer := 380;

-- Centered goal in 800-wide screen
constant GOAL_WIDTH     : integer := 280;
constant GOAL_LEFT_X    : integer := SCREEN_W/2 - GOAL_WIDTH/2;  -- 260
constant GOAL_RIGHT_X   : integer := SCREEN_W/2 + GOAL_WIDTH/2;  -- 540

-- Goalie geometry (wider rectangle)
constant GOALIE_W       : integer := 80;
constant GOALIE_H       : integer := 20;
constant GOALIE_Y       : integer := GOAL_LINE_Y + 20;
constant GOALIE_SPEED   : integer := 3;

-- Pre-computed center and left/right target positions
```

```
constant GOAL_CENTER_X  : integer := (GOAL_LEFT_X + GOAL_RIGHT_X)/2;
constant GOAL_HALF_SPAN : integer := (GOAL_RIGHT_X - GOAL_LEFT_X)/2;
-- left / center / right positions are more central (harder shots)
constant GOAL_LEFT_POS_X  : integer := GOAL_CENTER_X - GOAL_HALF_SPAN/2;
constant GOAL_RIGHT_POS_X : integer := GOAL_CENTER_X + GOAL_HALF_SPAN/2;

signal ball_x, ball_y   : integer := SCREEN_W/2;
signal aim_x,  aim_y    : integer := SCREEN_W/2;
signal vx, vy           : integer := 0;

signal goalie_x         : integer := GOAL_CENTER_X;
signal goalie_target_x  : integer := GOAL_CENTER_X;

signal result_good      : std_logic := '0';
signal result_timer     : integer range 0 to 120 := 0;

-- which dive was used last shot: 0=left,1=center,2=right
signal goalie_last_dive : integer range 0 to 2 := 1;

----------------------------------------------------------------------
-- Power meter parameters (oscillating line)
----------------------------------------------------------------------
constant POWER_BAR_W  : integer := 220;
constant POWER_BAR_H  : integer := 8;
constant POWER_BAR_Y  : integer := SCREEN_H - 40;
constant POWER_BAR_X0 : integer := SCREEN_W/2 - POWER_BAR_W/2;

signal power_pos : integer range 0 to POWER_BAR_W := 0;  -- 0..W
signal power_dir : integer := 1;                         -- +1 or -1

----------------------------------------------------------------------
-- Tiny LFSR for pseudo-random goalie choice
----------------------------------------------------------------------
signal lfsr : unsigned(7 downto 0) := "10101010";

----------------------------------------------------------------------
-- 60-second game timer + high score
----------------------------------------------------------------------
constant GAME_DURATION  : integer := 60;
signal time_left        : integer range 0 to GAME_DURATION := GAME_DURATION;
signal score            : integer range 0 to 99 := 0;
signal high_score       : integer range 0 to 99 := 0;

----------------------------------------------------------------------
```

```
    -- 7-seg score to display (current or high)
    ------------------------------------------------------------------
    signal display_score : integer range 0 to 99 := 0;
    signal ones, tens    : integer range 0 to 9;

    ------------------------------------------------------------------
    -- 7-segment driver
    ------------------------------------------------------------------
    signal refresh_cnt : unsigned(15 downto 0) := (others => '0');
    signal digit_sel   : unsigned(2 downto 0);
    signal digit_value : integer range 0 to 9 := 0;
    signal seg_i       : std_logic_vector(6 downto 0);
    signal an_i        : std_logic_vector(7 downto 0);

    ------------------------------------------------------------------
    -- RGB from renderer
    ------------------------------------------------------------------
    signal r_i, g_i, b_i : std_logic_vector(3 downto 0);

begin
    ------------------------------------------------------------------
    -- Clock divider: 100 MHz -> 25 MHz
    ------------------------------------------------------------------
    process(clk100)
    begin
        if rising_edge(clk100) then
            clk_div <= clk_div + 1;
        end if;
    end process;

    clk25 <= clk_div(1);

    ------------------------------------------------------------------
    -- Button synchronisers
    ------------------------------------------------------------------
    process(clk25, reset_n)
    begin
        if reset_n = '0' then
            btnL_sync <= '0';
            btnR_sync <= '0';
            btnU_sync <= '0';
            btnD_sync <= '0';
            btnC_sync <= '0';
        elsif rising_edge(clk25) then
```

```
        btnL_sync <= btnL;
        btnR_sync <= btnR;
        btnU_sync <= btnU;
        btnD_sync <= btnD;
        btnC_sync <= btnC;
    end if;
end process;

-----------------------------------------------------------------
-- LFSR
-----------------------------------------------------------------
process(clk25, reset_n)
begin
    if reset_n = '0' then
        lfsr <= "10101010";
    elsif rising_edge(clk25) then
        lfsr <= lfsr(6 downto 0) & (lfsr(7) xor lfsr(5));
    end if;
end process;

-----------------------------------------------------------------
-- VGA sync block (external)
-----------------------------------------------------------------
vga_core : entity work.vga_sync
    port map (
        pixel_clk => clk25,
        red_in    => '1',
        green_in  => '1',
        blue_in   => '1',
        red_out   => open,
        green_out => open,
        blue_out  => open,
        hsync     => hs,
        vsync     => vs,
        pixel_row => pixel_row_sv,
        pixel_col => pixel_col_sv
    );

vga_hs <= hs;
vga_vs <= vs;

pixel_x  <= unsigned(pixel_col_sv);
pixel_y  <= unsigned(pixel_row_sv);
video_on <= '1' when (to_integer(pixel_x) < SCREEN_W and
```

```
                    to_integer(pixel_y) < SCREEN_H) else '0';

-----------------------------------------------------------------
-- Frame tick from vsync edge
-----------------------------------------------------------------
process(clk25)
begin
    if rising_edge(clk25) then
        vs_d <= vs;
        if vs_d = '0' and vs = '1' then
            frame_tick <= '1';
        else
            frame_tick <= '0';
        end if;
    end if;
end process;

-----------------------------------------------------------------
-- 1-second tick generator
-----------------------------------------------------------------
process(clk25, reset_n)
begin
    if reset_n = '0' then
        sec_cnt      <= (others => '0');
        one_sec_tick <= '0';
    elsif rising_edge(clk25) then
        if sec_cnt = SEC_MAX then
            sec_cnt      <= (others => '0');
            one_sec_tick <= '1';
        else
            sec_cnt      <= sec_cnt + 1;
            one_sec_tick <= '0';
        end if;
    end if;
end process;

-----------------------------------------------------------------
-- Game timer and high score
-----------------------------------------------------------------
process(clk25, reset_n)
begin
    if reset_n = '0' then
        time_left  <= GAME_DURATION;
        high_score <= 0;
```

```
    elsif rising_edge(clk25) then
        if one_sec_tick = '1' then
            if time_left > 0 then
                if time_left = 1 then
                    if score > high_score then
                        high_score <= score;
                    end if;
                end if;
                time_left <= time_left - 1;
            end if;
        end if;
    end if;
end process;

-----------------------------------------------------------------------
-- Game logic
-----------------------------------------------------------------------
game_proc : process(clk25, reset_n)
    variable aim_step    : integer := 6;
    variable ax          : integer;
    variable power_speed : integer;
    variable new_dive    : integer;
begin
    if reset_n = '0' then
        game_state      <= AIM;
        ball_x          <= SCREEN_W/2;
        ball_y          <= PEN_SPOT_Y;
        aim_x           <= SCREEN_W/2;
        aim_y           <= GOAL_LINE_Y + 40;
        vx              <= 0;
        vy              <= 0;
        goalie_x        <= GOAL_CENTER_X;
        goalie_target_x <= GOAL_CENTER_X;
        goalie_last_dive<= 1;
        result_good     <= '0';
        result_timer    <= 0;
        score           <= 0;
        power_pos       <= 0;
        power_dir       <= 1;
    elsif rising_edge(clk25) then
        if frame_tick = '1' then

            -- Time up -> GAMEOVER
            if time_left = 0 then
```

```
    game_state <= GAMEOVER;
end if;

-- Goalie movement
if game_state = AIM then
    goalie_x        <= GOAL_CENTER_X;
    goalie_target_x <= GOAL_CENTER_X;
elsif game_state = SHOT then
    -- move towards target, clamp to avoid vibrating
    if goalie_x < goalie_target_x then
        if goalie_x + GOALIE_SPEED >= goalie_target_x then
            goalie_x <= goalie_target_x;
        else
            goalie_x <= goalie_x + GOALIE_SPEED;
        end if;
    elsif goalie_x > goalie_target_x then
        if goalie_x - GOALIE_SPEED <= goalie_target_x then
            goalie_x <= goalie_target_x;
        else
            goalie_x <= goalie_x - GOALIE_SPEED;
        end if;
    end if;
end if;

-- Power meter oscillation (only in AIM)
if game_state = AIM then
    if power_dir = 1 then
        if power_pos >= POWER_BAR_W then
            power_pos <= POWER_BAR_W;
            power_dir <= -1;
        else
            power_pos <= power_pos + 1;
        end if;
    else
        if power_pos <= 0 then
            power_pos <= 0;
            power_dir <= 1;
        else
            power_pos <= power_pos - 1;
        end if;
    end if;
end if;

-- Main state machine
```

```
case game_state is
    ------------------------------------------------------------------
    when AIM =>
        ball_x <= SCREEN_W/2;
        ball_y <= PEN_SPOT_Y;

        -- aiming movement
        if btnL_sync = '1' then
            aim_x <= aim_x - aim_step;
        elsif btnR_sync = '1' then
            aim_x <= aim_x + aim_step;
        end if;

        if btnU_sync = '1' then
            aim_y <= aim_y - aim_step;
        elsif btnD_sync = '1' then
            aim_y <= aim_y + aim_step;
        end if;

        -- clamp aim inside goal region
        if aim_x < GOAL_LEFT_X + 20 then
            aim_x <= GOAL_LEFT_X + 20;
        elsif aim_x > GOAL_RIGHT_X - 20 then
            aim_x <= GOAL_RIGHT_X - 20;
        end if;

        if aim_y < GOAL_LINE_Y + 10 then
            aim_y <= GOAL_LINE_Y + 10;
        elsif aim_y > GOAL_LINE_Y + 120 then
            aim_y <= GOAL_LINE_Y + 120;
        end if;

        -- shoot with center button
        if btnC_sync = '1' then
            ------------------------------------------------------------------
            -- Choose goalie dive: 0=left,1=center,2=right
            -- Use LFSR bits, but don't repeat last_dive
            ------------------------------------------------------------------
            new_dive := to_integer(unsigned(lfsr(1 downto 0))) mod 3;
            if new_dive = goalie_last_dive then
                new_dive := (new_dive + 1) mod 3;
            end if;
            goalie_last_dive <= new_dive;
```

```
        case new_dive is
           when 0 =>
              goalie_target_x <= GOAL_LEFT_POS_X;
           when 1 =>
              goalie_target_x <= GOAL_CENTER_X;
           when others =>
              goalie_target_x <= GOAL_RIGHT_POS_X;
        end case;

        goalie_x <= GOAL_CENTER_X;  -- start dive from center

        -- horizontal ball velocity (gentler so shots stay in)
        ax := aim_x - (SCREEN_W/2);
        if    ax < -80 then vx <= -2;
        elsif ax < -30 then vx <= -1;
        elsif ax >  80 then vx <=  2;
        elsif ax >  30 then vx <=  1;
        else                vx <=  0;
        end if;

        -- vertical speed from power meter (more levels)
        -- map 0..POWER_BAR_W -> speed 3..10
        power_speed := 3 + (power_pos * 7) / POWER_BAR_W;
        vy          <= -power_speed;

        game_state  <= SHOT;
     end if;

-----------------------------------------------------------------
when SHOT =>
     ball_x <= ball_x + vx;
     ball_y <= ball_y + vy;

     if (ball_y <= GOAL_LINE_Y + BALL_R) or (ball_y < 0) then
        if (ball_x > GOAL_LEFT_X + BALL_R) and
          (ball_x < GOAL_RIGHT_X - BALL_R) then
           -- collision with goalie rectangle (circle vs box)
           if (abs(ball_x - goalie_x) <= (GOALIE_W/2 + BALL_R)) and
              (ball_y >= GOALIE_Y - GOALIE_H/2 - BALL_R) and
              (ball_y <= GOALIE_Y + GOALIE_H/2 + BALL_R) then
              result_good <= '0';    -- saved
           else
              result_good <= '1';    -- GOAL
              if score < 99 then
```

```
                        score <= score + 1;
                    end if;
                end if;
            else
                result_good <= '0';        -- wide
            end if;

            result_timer <= 40;
            game_state   <= RESULT;
        end if;

    ---------------------------------------------------------------
    when RESULT =>
        if result_timer > 0 then
            result_timer <= result_timer - 1;
        else
            if time_left > 0 then
                ball_x     <= SCREEN_W/2;
                ball_y     <= PEN_SPOT_Y;
                aim_x      <= SCREEN_W/2;
                aim_y      <= GOAL_LINE_Y + 40;
                game_state <= AIM;
            else
                game_state <= GAMEOVER;
            end if;
        end if;

    ---------------------------------------------------------------
    when GAMEOVER =>
        null;
    end case;
  end if;
end if;
end process;

-----------------------------------------------------------------
-- Renderer: field, goal, goalie, ball, crosshair, power bar, text
-----------------------------------------------------------------
renderer : process(pixel_x, pixel_y, video_on,
                   ball_x, ball_y,
                   aim_x, aim_y,
                   goalie_x, game_state, result_good, power_pos)
  variable x, y : integer;
  variable dx, dy : integer;
```

```
    variable ball_on, goalie_on, net_on, field_on, aim_on : boolean;
    variable line_on : boolean := false; -- kept for completeness

    -- power meter
    variable power_bar_on  : boolean;
    variable power_line_on : boolean;
    variable line_x        : integer;

    -- text
    variable msg_x0, msg_y0 : integer;
    variable rel_x, rel_y    : integer;
    variable txt_on, outline_on : boolean;

    constant MSG_TOTAL_W : integer :=
        TXT_MSG_LEN*TXT_CHAR_W + (TXT_MSG_LEN-1)*TXT_GAP;
begin
    x := to_integer(pixel_x);
    y := to_integer(pixel_y);

    ball_on       := false;
    goalie_on     := false;
    net_on        := false;
    field_on      := false;
    aim_on        := false;
    power_bar_on  := false;
    power_line_on := false;
    txt_on        := false;
    outline_on    := false;

    if (x < SCREEN_W) and (y < SCREEN_H) then
        -- field: everything below goal line
        if y >= GOAL_LINE_Y then
            field_on := true;
        end if;

        -- goal frame (top and posts)
        if (y < GOAL_LINE_Y + 2) and (y >= GOAL_LINE_Y - 2) and
           (x >= GOAL_LEFT_X) and (x <= GOAL_RIGHT_X) then
            net_on := true;
        end if;
        if (x >= GOAL_LEFT_X-2 and x <= GOAL_LEFT_X+2 and y < GOAL_LINE_Y) or
           (x >= GOAL_RIGHT_X-2 and x <= GOAL_RIGHT_X+2 and y < GOAL_LINE_Y) then
            net_on := true;
        end if;
```

```
-- goalie rectangle
if (x >= goalie_x - GOALIE_W/2) and (x <= goalie_x + GOALIE_W/2) and
   (y >= GOALIE_Y - GOALIE_H/2) and (y <= GOALIE_Y + GOALIE_H/2) then
    goalie_on := true;
end if;

-- ball (simple circle)
dx := x - ball_x;
dy := y - ball_y;
if (dx*dx + dy*dy) <= BALL_R*BALL_R then
    ball_on := true;
end if;

-- aim crosshair (only in AIM state)
if game_state = AIM then
    if (abs(x-aim_x) < 6 and abs(y-aim_y) < 1) or
       (abs(y-aim_y) < 6 and abs(x-aim_x) < 1) then
        aim_on := true;
    end if;
end if;

-- POWER METER BAR (always visible)
line_x := POWER_BAR_X0 + power_pos;

if (y >= POWER_BAR_Y-POWER_BAR_H/2) and
   (y <= POWER_BAR_Y+POWER_BAR_H/2) and
   (x >= POWER_BAR_X0) and
   (x <= POWER_BAR_X0 + POWER_BAR_W) then
    power_bar_on := true;
end if;

if (y >= POWER_BAR_Y-POWER_BAR_H) and
   (y <= POWER_BAR_Y+POWER_BAR_H) and
   (abs(x - line_x) < 2) then
    power_line_on := true;
end if;

-- Comic text in RESULT state
if game_state = RESULT then
    msg_x0 := (SCREEN_W - MSG_TOTAL_W) / 2;
    msg_y0 := 180;
    rel_x  := x - msg_x0;
    rel_y  := y - msg_y0;
```

```
        txt_on     := text_fill_pixel   (result_good, rel_x, rel_y);
        outline_on := text_outline_pixel(result_good, rel_x, rel_y) and (not txt_on);
    end if;
end if;

-- COLORS
if video_on = '0' then
    r_i <= (others => '0');
    g_i <= (others => '0');
    b_i <= (others => '0');
else
    -- default sky
    r_i <= "0011";
    g_i <= "0111";
    b_i <= "1111";

    if field_on then
        r_i <= "0000";
        g_i <= "1001";  -- green
        b_i <= "0001";
    end if;

    if net_on then
        r_i <= "1111";
        g_i <= "1111";
        b_i <= "1111";
    end if;

    if goalie_on then
        r_i <= "0000";
        g_i <= "0000";
        b_i <= "1111";
    end if;

    if ball_on then
        r_i <= "1111";
        g_i <= "1111";
        b_i <= "1111";
    end if;

    if aim_on then
        r_i <= "1111";
        g_i <= "0000";
        b_i <= "0000";
```

```
        end if;

        -- power bar background
        if power_bar_on then
            r_i <= "0011";
            g_i <= "0011";
            b_i <= "0011";
        end if;

        -- power line color: red -> yellow -> green
        if power_line_on then
            if power_pos < POWER_BAR_W/3 then       -- low power: red
                r_i <= "1111";
                g_i <= "0000";
                b_i <= "0000";
            elsif power_pos < (2*POWER_BAR_W)/3 then -- mid: yellow
                r_i <= "1111";
                g_i <= "1111";
                b_i <= "0000";
            else                                    -- high: green
                r_i <= "0000";
                g_i <= "1111";
                b_i <= "0000";
            end if;
        end if;

        -- comic text on top
        if outline_on then
            r_i <= "0000";
            g_i <= "0000";
            b_i <= "0000";
        end if;

        if txt_on then
            r_i <= "1111";
            g_i <= "1111";
            b_i <= "1111";
        end if;
    end if;
end process;

vga_r <= r_i;
vga_g <= g_i;
vga_b <= b_i;
```

```
-----------------------------------------------------------------
-- Which score to show? (time running: score, time over: high_score)
-----------------------------------------------------------------
process(score, high_score, time_left)
begin
   if time_left = 0 then
      display_score <= high_score;
   else
      display_score <= score;
   end if;

   tens <= display_score / 10;
   ones <= display_score mod 10;
end process;

-----------------------------------------------------------------
-- 7-segment refresh counter
-----------------------------------------------------------------
refresh_clk : process(clk100, reset_n)
begin
   if reset_n = '0' then
      refresh_cnt <= (others => '0');
   elsif rising_edge(clk100) then
      refresh_cnt <= refresh_cnt + 1;
   end if;
end process;

digit_sel <= refresh_cnt(15 downto 13);

-----------------------------------------------------------------
-- Digit multiplexing (2 rightmost digits)
-----------------------------------------------------------------
mux_proc : process(digit_sel, ones, tens)
begin
   an_i       <= "11111111";
   digit_value <= 0;

   case digit_sel is
      when "000" =>
         an_i       <= "11111110"; -- digit 0 (rightmost)
         digit_value <= ones;
      when "001" =>
         an_i       <= "11111101"; -- digit 1
```

```
            digit_value <= tens;
         when others =>
            an_i        <= "11111111";
      end case;
   end process;

   ----------------------------------------------------------------------
   -- 7-seg decoder (0-9, active-low, seg = g f e d c b a)
   ----------------------------------------------------------------------
   seg_decode : process(digit_value)
   begin
      case digit_value is
         when 0 => seg_i <= "1000000"; -- 0
         when 1 => seg_i <= "1111001"; -- 1
         when 2 => seg_i <= "0100100"; -- 2
         when 3 => seg_i <= "0110000"; -- 3
         when 4 => seg_i <= "0011001"; -- 4
         when 5 => seg_i <= "0010010"; -- 5
         when 6 => seg_i <= "0000010"; -- 6
         when 7 => seg_i <= "1111000"; -- 7
         when 8 => seg_i <= "0000000"; -- 8
         when 9 => seg_i <= "0010000"; -- 9
         when others => seg_i <= "1111111"; -- all off
      end case;
   end process;

   an  <= an_i;
   seg <= seg_i;

end architecture;
```