```
----------------------------------------------------------------------------------
-- Engineer: Zachary Montoya
-- Submitted Date: 12-15-22
-- Module Name:    Top - Behavioral
-- Project Name:   Asteroids
-- Target Devices: Zybo Z7-10
-- Tool versions: Vivado 2020.2
--
-- Comment:
--         This file was entirely created the engineer listed above.
--
----------------------------------------------------------------------------------

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

entity missile is
   generic(
      missile_velocity: integer := 1
   );

   port(
      clk, reset: in std_logic;
      btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
      video_on: in std_logic;
      pixel_x, pixel_y: in std_logic_vector(9 downto 0);
      missile_on_out: out std_logic;
      missile_rgb_out: out std_logic_vector(2 downto 0);
      rom_selector_out: in std_logic_vector(2 downto 0);
      bar_x_reg_out, bar_y_reg_out: in unsigned( 9 downto 0);
      fire_out,fire_btn_signal_out: in std_logic;
      fire_ready_out: out std_logic;
      missile_left_out, missile_right_out, missile_top_out, missile_bottom_out: out
unsigned(9 downto 0)

   );
end missile;

--
================================================================================================
--
================================================================================================

architecture rtl of missile is

-- Signal used to control speed of ball and how often pushbuttons are checked for
paddle movement.
   signal refr_tick: std_logic;

-- x, y coordinates (0,0 to (639, 479)
   signal pix_x, pix_y: unsigned(9 downto 0);

   -- WALL1 - LEFT
   constant WALL1_X_L: integer := 0;
   constant WALL1_X_R: integer := 20;

   -- WALL2 - RIGHT
   constant WALL2_X_L: integer := 619;
   constant WALL2_X_R: integer := 639;

   -- WALL3 - BOTTOM
   constant WALL3_X_T: integer := 409;
   constant WALL3_X_B: integer := 479;

   -- WALL4 - TOP
   constant WALL4_X_T: integer := 0;
   constant WALL4_X_B: integer := 20;

```

```
-- Square ball -- ball left, right, top and bottom all vary. Left and top driven by registers below.
   constant BALL_SIZE: integer := 16; --------------------CHANGED Increased to 16
   signal ball_x_l, ball_x_r: unsigned(9 downto 0);
   signal ball_y_t, ball_y_b: unsigned(9 downto 0);

-- Reg to track left and top boundary
   signal ball_x_reg, ball_x_next, ball_x_next_mux: unsigned(9 downto 0);
   signal ball_y_reg, ball_y_next, ball_y_next_mux: unsigned(9 downto 0);

-- reg to track ball speed
   signal x_delta_reg, x_delta_next: unsigned(9 downto 0);
   signal y_delta_reg, y_delta_next: unsigned(9 downto 0);

-- ball movement can be pos or neg
   constant BALL_V_P: unsigned(9 downto 0):= to_unsigned(missile_velocity,10);
   constant BALL_V_N: unsigned(9 downto 0):= unsigned(to_signed(-missile_velocity,10));
   constant BALL_V_NULL: unsigned(9 downto 0):= to_unsigned(0,10);

-- round ball image
   type rom_type is array(0 to 15) of std_logic_vector(0 to 15); ----------Changed from array(0 to 7)
   constant BALL_ROM_UP: rom_type:= (
      "0000000110000000",
      "1100000110000011",
      "1100000000000011",
      "0000000000000000",
      "0000000000000000",
      "0000000110000000",
      "1100000110000011",
      "1100000110000011",
      "1100000110000011",
      "1100000110000011",
      "1100000110000011",
      "1100000000000011",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000");
   constant BALL_ROM_DOWN: rom_type:= (
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "1100000110000011",
      "1100000000000011",
      "1100000000000011",
      "1100000000000011",
      "1100000000000011",
      "1100000000000011",
      "0000000110000000",
      "0000000000000000",
      "0000000000000000",
      "1100000000000011",
      "1100000110000011",
      "0000000110000000");
   constant BALL_ROM_LEFT: rom_type:= (
      "0110001111110000",
      "0110001111110000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "1100011111000000",
      "1100011111000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0110001111110000",
      "0110001111110000");
   constant BALL_ROM_RIGHT: rom_type:= (
      "0000111111100110",
      "0000111111100110",
      "0000000000000000",
      "0000000000000000",
```

```
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000011111100011",
      "0000011111100011",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000000000000000",
      "0000111111000110",
      "0000111111000110");
      -- Testing
      -- 0 is to paint the background and 1 will be to use the ball
      -- Can make more than one bit to make characters and other sprites.


   -- ball is 8x8, the address only needs to be 3 bits then.
   -- data will first be read as a row
   -- rom_bit will go to the bit in the row
   signal rom_addr, rom_col: unsigned(3 downto 0); ------------------------Changed
to 4 bits from unsigned (2 downto 0)
   signal rom_data: std_logic_vector(15 downto 0); ------------------------Changed
from (7 downto 0)
   signal rom_bit: std_logic;

-- object output signals -- new signal to indicate if scan coord is within ball
   signal sq_ball_on, rd_ball_on: std_logic;
   signal ball_rgb: std_logic_vector(2 downto 0);
   signal fire_ready_reg, fire_ready_next: std_logic;
   signal rom_selector_reg, rom_selector_next: std_logic_vector(2 downto 0);


--
================================================================================
=================
   begin

   missile_on_out <= rd_ball_on;
   missile_rgb_out <= ball_rgb;
   fire_ready_out <= fire_ready_next;


   with fire_out select
      ball_x_next_mux <= bar_x_reg_out when '0',
                         ball_x_next when '1';

   with fire_out select
      ball_y_next_mux <= bar_y_reg_out when '0',
                         ball_y_next when '1';

   process (clk, reset)
      begin
      if (reset = '1') then
         ball_x_reg <= bar_x_reg_out;
         ball_y_reg <= bar_y_reg_out;
         x_delta_reg <= ("0000000000");
         y_delta_reg <= ("0000000000");
         fire_ready_reg <= '0';
         rom_selector_reg <= "111";
      elsif (clk'event and clk = '1') then
         ball_x_reg <= ball_x_next_mux;
         ball_y_reg <= ball_y_next_mux;
         x_delta_reg <= x_delta_next;
         y_delta_reg <= y_delta_next;
         fire_ready_reg <= fire_ready_next;
         rom_selector_reg <= rom_selector_next;
      end if;
   end process;

--
================================================================================
=================
   pix_x <= unsigned(pixel_x);
   pix_y <= unsigned(pixel_y);

-- Refr_tick: 1-clock tick asserted at start of v_sync, e.g., when the screen is
refreshed -- speed is 60 Hz
   -- refr_tick <= '1' when (pix_y = 1) and (pix_x = 1) else '0';
```

```
   refr_tick <= '1' when (((pix_y = 1) or (pix_y = 73) or (pix_y = 146) or (pix_y = 220) or (pix_y = 293) or (pix_y = 365)) and (pix_x = 1)) else '0';
   -- refr_tick <= '1' when (pix_y = 1) or (pix_y = 240) else '0';
--
================================================================================
================

   process(rom_selector_reg, rom_selector_out)
      begin
      rom_selector_next <= rom_selector_reg;
      if (btn(0) = '1' and fire_btn_signal_out = '0') then --Add fire_button_ready_signal!!
         rom_selector_next <= rom_selector_out;
      end if;
   end process;

--
================================================================================
================
-- set coordinates of square ball.
   ball_x_l <= ball_x_reg;
   ball_y_t <= ball_y_reg;
   ball_x_r <= ball_x_l + BALL_SIZE - 1;
   ball_y_b <= ball_y_t + BALL_SIZE - 1;

-- pixel within square ball
   sq_ball_on <= '1' when (ball_x_l <= pix_x) and (pix_x <= ball_x_r) and (ball_y_t <= pix_y) and (pix_y <= ball_y_b) else '0';

-- Map scan coord to ROM addr/col -- use low order three bits of pixel and ball positions. ROM row
   rom_addr <= pix_y(3 downto 0) - ball_y_t(3 downto 0); -------------------- CHANGED TO 4 BITS

-- ROM column
   rom_col <= pix_x(3 downto 0) - ball_x_l(3 downto 0);-------------------- CHANGED TO 4 BITS

-- Get row data
   process(fire_out,rom_selector_out)
      begin
      if (fire_out = '0') then
         if (rom_selector_out = "110") then
            rom_data <= BALL_ROM_UP(to_integer(rom_addr));
         elsif (rom_selector_out = "000") then
            rom_data <= BALL_ROM_DOWN(to_integer(rom_addr));
         elsif (rom_selector_out = "100") then
            rom_data <= BALL_ROM_LEFT(to_integer(rom_addr));
         elsif (rom_selector_out = "010") then
            rom_data <= BALL_ROM_RIGHT(to_integer(rom_addr));
         end if;
      elsif (fire_out = '1') then
         if (rom_selector_reg = "110") then
            rom_data <= BALL_ROM_UP(to_integer(rom_addr));
         elsif (rom_selector_reg = "000") then
            rom_data <= BALL_ROM_DOWN(to_integer(rom_addr));
         elsif (rom_selector_reg = "100") then
            rom_data <= BALL_ROM_LEFT(to_integer(rom_addr));
         elsif (rom_selector_reg = "010") then
            rom_data <= BALL_ROM_RIGHT(to_integer(rom_addr));
         end if;
      end if;
   end process;


-- Get column bit
   rom_bit <= rom_data(to_integer(rom_col));

-- Turn ball on only if within square and the ROM bit is 1.
   rd_ball_on <= '1' when (sq_ball_on = '1') and (rom_bit = '1') else '0';
   ball_rgb <= "111"; -- WHITE BALL COLOR

-- Update the ball position 60 times per second.
   ball_x_next <= ball_x_reg + x_delta_reg when refr_tick = '1' else ball_x_reg;
   ball_y_next <= ball_y_reg + y_delta_reg when refr_tick = '1' else ball_y_reg;


-- Set the value of the next ball position according to the boundaries.
```

```
   -- process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_b,
fire_ready_reg, rom_selector_reg)
   --    begin
   --    x_delta_next <= x_delta_reg; -- don't move
   --    y_delta_next <= y_delta_reg; -- don't move
   --    fire_ready_next <= fire_ready_reg;

   --    if (rom_selector_reg = "110" and ball_y_t >= WALL4_X_B and ball_y_t < 479 )
then -- DOWN
   --          y_delta_next <= BALL_V_N;
   --          x_delta_next <= BALL_V_NULL;
   --    elsif (rom_selector_reg = "110" and ball_y_t = WALL4_X_B and ball_y_t < 479
) then -- DOWN
   --          y_delta_next <= BALL_V_NULL;
   --          x_delta_next <= BALL_V_NULL;
   --          fire_ready_next <= '1';

   --    elsif (rom_selector_reg = "000" and ball_y_b <= WALL3_X_T and ball_y_b > 0)
then -- UP
   --          y_delta_next <= BALL_V_P;
   --          x_delta_next <= BALL_V_NULL;
   --    elsif (rom_selector_reg = "000" and ball_y_b = WALL3_X_T and ball_y_b > 0)
then -- UP
   --          y_delta_next <= BALL_V_NULL;
   --          x_delta_next <= BALL_V_NULL;
   --          fire_ready_next <= '1';

   --    elsif (rom_selector_reg = "010" and ball_x_l >= WALL1_X_R and ball_x_l > 0)
then -- RIGHT
   --          x_delta_next <= BALL_V_N;
   --          y_delta_next <= BALL_V_NULL;
   --    elsif (rom_selector_reg = "010" and ball_x_l = WALL1_X_R and ball_x_l > 0)
then -- RIGHT
   --             y_delta_next <= BALL_V_NULL;
   --             x_delta_next <= BALL_V_NULL;
   --             fire_ready_next <= '1';

   --    elsif (rom_selector_reg = "100" and ball_x_r <= WALL2_X_L and ball_x_r <
639) then -- LEFT
   --          x_delta_next <= BALL_V_P;
   --          y_delta_next <= BALL_V_NULL;
   --    elsif (rom_selector_reg = "100" and ball_x_r = WALL2_X_L and ball_x_r < 639)
then -- LEFT
   --          y_delta_next <= BALL_V_NULL;
   --          x_delta_next <= BALL_V_NULL;
   --          fire_ready_next <= '1';
   --    end if ;

      -- Set the value of the next ball position according to the boundaries.
      process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_b,
fire_ready_reg, rom_selector_reg)
      begin
      x_delta_next <= x_delta_reg; -- don't move
      y_delta_next <= y_delta_reg; -- don't move
      fire_ready_next <= fire_ready_reg;

      if (rom_selector_reg = "110" and ball_y_t >= WALL4_X_B and ball_y_t < 479 )
then -- DOWN
            y_delta_next <= BALL_V_N;
            x_delta_next <= BALL_V_NULL;

            if ( ball_y_t <= WALL4_X_B ) then
               y_delta_next <= BALL_V_NULL;
               x_delta_next <= BALL_V_NULL;
               fire_ready_next <= '1';
            else
               fire_ready_next <= '0';
            end if;

      elsif (rom_selector_reg = "000" and ball_y_b <= WALL3_X_T and ball_y_b > 0)
then -- UP
            y_delta_next <= BALL_V_P;
            x_delta_next <= BALL_V_NULL;

            if (ball_y_b >= WALL3_X_T) then
               y_delta_next <= BALL_V_NULL;
               x_delta_next <= BALL_V_NULL;
               fire_ready_next <= '1';
            else
```

```
               fire_ready_next <= '0';
            end if;

      elsif (rom_selector_reg = "010" and ball_x_l >= WALL1_X_R and ball_x_l > 0)
then -- RIGHT
            x_delta_next <= BALL_V_N;
            y_delta_next <= BALL_V_NULL;

            if (ball_x_l <= WALL1_X_R ) then
               y_delta_next <= BALL_V_NULL;
               x_delta_next <= BALL_V_NULL;
               fire_ready_next <= '1';
            else
               fire_ready_next <= '0';
            end if;

      elsif (rom_selector_reg = "100" and ball_x_r <= WALL2_X_L and ball_x_r < 639)
then -- LEFT
            x_delta_next <= BALL_V_P;
            y_delta_next <= BALL_V_NULL;

            if (ball_x_r >= WALL2_X_L) then
               y_delta_next <= BALL_V_NULL;
               x_delta_next <= BALL_V_NULL;
               fire_ready_next <= '1';
            else
               fire_ready_next <= '0';
            end if;
      end if ;

   end process;

--
================================================================================
=================


end rtl;



-- -- ========================
```