Christopher Spadavecchia
Eli Shtindler
CPE 487

# Lab 6

## Project pong:

[Original Game]

As shown in the video, after uploading the original code, you could move the paddle in the pong game using the BTNL and BNTR. The original code does not keep track of the score/hits, has the same bat width throughout the game, and the ball speed does not change.

[Modified game with a counter hits, changed bat width, and changed ball speed]

After modifying the code, the original game is still playable but there are some modifications. Now, the score/hits are being tracked on the seven segment display on the FPGA. Also, the bat width becomes smaller and the ball speed increases after each hit.

## bat_n_ball.vhd

Original:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY bat_n_ball IS
    PORT (
        v_sync : IN STD_LOGIC;
        pixel_row : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
        pixel_col : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
        bat_x : IN STD_LOGIC_VECTOR (10 DOWNTO 0); -- current bat x position
        serve : IN STD_LOGIC; -- initiates serve
        red : OUT STD_LOGIC;
        green : OUT STD_LOGIC;
        blue : OUT STD_LOGIC
    );
END bat_n_ball;

ARCHITECTURE Behavioral OF bat_n_ball IS
    CONSTANT bsize : INTEGER := 8; -- ball size in pixels
    CONSTANT bat_w : INTEGER := 20; -- bat width in pixels
    CONSTANT bat_h : INTEGER := 3; -- bat height in pixels
    -- distance ball moves each frame
    CONSTANT ball_speed : STD_LOGIC_VECTOR (10 DOWNTO 0) := CONV_STD_LOGIC_VECTOR (6, 11);
    SIGNAL ball_on : STD_LOGIC; -- indicates whether ball is at current pixel position
    SIGNAL bat_on : STD_LOGIC; -- indicates whether bat at over current pixel position
    SIGNAL game_on : STD_LOGIC := '0'; -- indicates whether ball is in play
```

```
    -- current ball position - intitialized to center of screen
    SIGNAL ball_x : STD_LOGIC_VECTOR(10 DOWNTO 0) := CONV_STD_LOGIC_VECTOR(400, 11);
    SIGNAL ball_y : STD_LOGIC_VECTOR(10 DOWNTO 0) := CONV_STD_LOGIC_VECTOR(300, 11);
    -- bat vertical position
    CONSTANT bat_y : STD_LOGIC_VECTOR(10 DOWNTO 0) := CONV_STD_LOGIC_VECTOR(500, 11);
    -- current ball motion - initialized to (+ ball_speed) pixels/frame in both X and Y directions
    SIGNAL ball_x_motion, ball_y_motion : STD_LOGIC_VECTOR(10 DOWNTO 0) := ball_speed;
BEGIN
    red <= NOT bat_on; -- color setup for red ball and cyan bat on white background
    green <= NOT ball_on;
    blue <= NOT ball_on;
    -- process to draw round ball
    -- set ball_on if current pixel address is covered by ball position
    balldraw : PROCESS (ball_x, ball_y, pixel_row, pixel_col) IS
        VARIABLE vx, vy : STD_LOGIC_VECTOR (10 DOWNTO 0); -- 9 downto 0
    BEGIN
        IF pixel_col <= ball_x THEN -- vx = |ball_x - pixel_col|
            vx := ball_x - pixel_col;
        ELSE
            vx := pixel_col - ball_x;
        END IF;
        IF pixel_row <= ball_y THEN -- vy = |ball_y - pixel_row|
            vy := ball_y - pixel_row;
        ELSE
            vy := pixel_row - ball_y;
        END IF;
        IF ((vx * vx) + (vy * vy)) < (bsize * bsize) THEN -- test if radial distance < bsize
            ball_on <= game_on;
        ELSE
            ball_on <= '0';
        END IF;
    END PROCESS;
    -- process to draw bat
    -- set bat_on if current pixel address is covered by bat position
    batdraw : PROCESS (bat_x, pixel_row, pixel_col) IS
        VARIABLE vx, vy : STD_LOGIC_VECTOR (10 DOWNTO 0); -- 9 downto 0
    BEGIN
        IF ((pixel_col >= bat_x - bat_w) OR (bat_x <= bat_w)) AND
         pixel_col <= bat_x + bat_w AND
             pixel_row >= bat_y - bat_h AND
             pixel_row <= bat_y + bat_h THEN
                bat_on <= '1';
        ELSE
            bat_on <= '0';
        END IF;
    END PROCESS;
    -- process to move ball once every frame (i.e., once every vsync pulse)
    mball : PROCESS
        VARIABLE temp : STD_LOGIC_VECTOR (11 DOWNTO 0);
    BEGIN
        WAIT UNTIL rising_edge(v_sync);
        IF serve = '1' AND game_on = '0' THEN -- test for new serve
            game_on <= '1';
            ball_y_motion <= (NOT ball_speed) + 1; -- set vspeed to (- ball_speed) pixels
        ELSIF ball_y <= bsize THEN -- bounce off top wall
            ball_y_motion <= ball_speed; -- set vspeed to (+ ball_speed) pixels
        ELSIF ball_y + bsize >= 600 THEN -- if ball meets bottom wall
            ball_y_motion <= (NOT ball_speed) + 1; -- set vspeed to (- ball_speed) pixels
            game_on <= '0'; -- and make ball disappear
        END IF;
        -- allow for bounce off left or right of screen
        IF ball_x + bsize >= 800 THEN -- bounce off right wall
            ball_x_motion <= (NOT ball_speed) + 1; -- set hspeed to (- ball_speed) pixels
        ELSIF ball_x <= bsize THEN -- bounce off left wall
```

```
            ball_x_motion <= ball_speed; -- set hspeed to (+ ball_speed) pixels
        END IF;
        -- allow for bounce off bat
        IF (ball_x + bsize/2) >= (bat_x - bat_w) AND
         (ball_x - bsize/2) <= (bat_x + bat_w) AND
             (ball_y + bsize/2) >= (bat_y - bat_h) AND
             (ball_y - bsize/2) <= (bat_y + bat_h) THEN
                ball_y_motion <= (NOT ball_speed) + 1; -- set vspeed to (- ball_speed) pixels
        END IF;
        -- compute next ball vertical position
        -- variable temp adds one more bit to calculation to fix unsigned underflow problems
        -- when ball_y is close to zero and ball_y_motion is negative
        temp := ('0' & ball_y) + (ball_y_motion(10) & ball_y_motion);
        IF game_on = '0' THEN
            ball_y <= CONV_STD_LOGIC_VECTOR(440, 11);
        ELSIF temp(11) = '1' THEN
            ball_y <= (OTHERS => '0');
        ELSE ball_y <= temp(10 DOWNTO 0); -- 9 downto 0
        END IF;
        -- compute next ball horizontal position
        -- variable temp adds one more bit to calculation to fix unsigned underflow problems
        -- when ball_x is close to zero and ball_x_motion is negative
        temp := ('0' & ball_x) + (ball_x_motion(10) & ball_x_motion);
        IF temp(11) = '1' THEN
            ball_x <= (OTHERS => '0');
        ELSE ball_x <= temp(10 DOWNTO 0);
        END IF;
    END PROCESS;
END Behavioral;
```

Modified:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY bat_n_ball IS
    PORT (
        v_sync : IN STD_LOGIC;
        pixel_row : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
        pixel_col : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
        bat_x : IN STD_LOGIC_VECTOR (10 DOWNTO 0); -- current bat x position
        serve : IN STD_LOGIC; -- initiates serve
        red : OUT STD_LOGIC;
        green : OUT STD_LOGIC;
        blue : OUT STD_LOGIC;
        score : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
    );
END bat_n_ball;

ARCHITECTURE Behavioral OF bat_n_ball IS
    CONSTANT bsize : INTEGER := 8; -- ball size in pixels
    CONSTANT bat_w : INTEGER := 40; -- bat width in pixels
    CONSTANT bat_h : INTEGER := 3; -- bat height in pixels
    -- distance ball moves each frame
    CONSTANT ball_speed : STD_LOGIC_VECTOR (10 DOWNTO 0) := CONV_STD_LOGIC_VECTOR (6, 11);
    SIGNAL ball_on : STD_LOGIC; -- indicates whether ball is at current pixel position
    SIGNAL bat_on : STD_LOGIC; -- indicates whether bat at over current pixel position
    SIGNAL game_on : STD_LOGIC := '0'; -- indicates whether ball is in play
    -- current ball position - intitialized to center of screen
    SIGNAL ball_x : STD_LOGIC_VECTOR(10 DOWNTO 0) := CONV_STD_LOGIC_VECTOR(400, 11);
    SIGNAL ball_y : STD_LOGIC_VECTOR(10 DOWNTO 0) := CONV_STD_LOGIC_VECTOR(300, 11);
    -- bat vertical position
    CONSTANT bat_y : STD_LOGIC_VECTOR(10 DOWNTO 0) := CONV_STD_LOGIC_VECTOR(500, 11);
    -- current ball motion - initialized to (+ ball_speed) pixels/frame in both X and Y directions
    SIGNAL ball_x_motion, ball_y_motion : STD_LOGIC_VECTOR(10 DOWNTO 0) := ball_speed;
    SIGNAL hits : STD_LOGIC_VECTOR(15 DOWNTO 0);
    SIGNAL hit_on : STD_LOGIC;
BEGIN
    score <= hits;
    red <= NOT bat_on; -- color setup for red ball and cyan bat on white background
    green <= NOT ball_on;
    blue <= NOT ball_on;
    -- process to draw round ball
    -- set ball_on if current pixel address is covered by ball position
    balldraw : PROCESS (ball_x, ball_y, pixel_row, pixel_col) IS
        VARIABLE vx, vy : STD_LOGIC_VECTOR (10 DOWNTO 0); -- 9 downto 0
    BEGIN
        IF pixel_col <= ball_x THEN -- vx = |ball_x - pixel_col|
            vx := ball_x - pixel_col;
        ELSE
            vx := pixel_col - ball_x;
        END IF;
        IF pixel_row <= ball_y THEN -- vy = |ball_y - pixel_row|
            vy := ball_y - pixel_row;
        ELSE
            vy := pixel_row - ball_y;
        END IF;
        IF ((vx * vx) + (vy * vy)) < (bsize * bsize) THEN -- test if radial distance < bsize
            ball_on <= game_on;
```

```
    ELSE
        ball_on <= '0';
    END IF;
END PROCESS;
-- process to draw bat
-- set bat_on if current pixel address is covered by bat position
batdraw : PROCESS (bat_x, pixel_row, pixel_col) IS
    VARIABLE vx, vy : STD_LOGIC_VECTOR (10 DOWNTO 0); -- 9 downto 0
BEGIN
    IF ((pixel_col >= bat_x - (bat_w - CONV_INTEGER(hits))) OR (bat_x <= (bat_w - CONV_INTEGER(hits)))) AND
     pixel_col <= bat_x + (bat_w - CONV_INTEGER(hits)) AND
         pixel_row >= bat_y - bat_h AND
         pixel_row <= bat_y + bat_h THEN
            bat_on <= '1';
    ELSE
        bat_on <= '0';
    END IF;
END PROCESS;
-- process to move ball once every frame (i.e., once every vsync pulse)
mball : PROCESS
    VARIABLE temp : STD_LOGIC_VECTOR (11 DOWNTO 0);
BEGIN
    WAIT UNTIL rising_edge(v_sync);
    IF serve = '1' AND game_on = '0' THEN -- test for new serve
        game_on <= '1';
        hit_on <= '1';
        hits <= CONV_STD_LOGIC_VECTOR(0, 16);
        ball_x_motion <= (NOT ball_speed) + 1;
        ball_y_motion <= (NOT ball_speed) + 1; -- set vspeed to (- ball_speed) pixels
    ELSIF ball_y <= bsize THEN -- bounce off top wall
        ball_y_motion <= (ball_speed + hits(10 DOWNTO 0)); -- set vspeed to (+ ball_speed) pixels
        hit_on <= '1';
    ELSIF ball_y + bsize >= 600 THEN -- if ball meets bottom wall
        ball_y_motion <= (NOT (ball_speed + hits(10 DOWNTO 0))) + 1; -- set vspeed to (- ball_speed) pixels
        game_on <= '0'; -- and make ball disappear
    END IF;
    -- allow for bounce off left or right of screen
    IF ball_x + bsize >= 800 THEN -- bounce off right wall
        ball_x_motion <= (NOT (ball_speed + hits(10 DOWNTO 0))) + 1; -- set hspeed to (- ball_speed) pixels
    ELSIF ball_x <= bsize THEN -- bounce off left wall
        ball_x_motion <= (ball_speed + hits(10 DOWNTO 0)); -- set hspeed to (+ ball_speed) pixels
    END IF;
    -- allow for bounce off bat
    IF (ball_x + bsize/2) >= (bat_x - (bat_w - CONV_INTEGER(hits))) AND
     (ball_x - bsize/2) <= (bat_x + (bat_w - CONV_INTEGER(hits))) AND
         (ball_y + bsize/2) >= (bat_y - bat_h) AND
         (ball_y - bsize/2) <= (bat_y + bat_h) THEN
            IF hit_on = '1' THEN
                hits <= hits + 1;
                hit_on <= '0';
            END IF;
            IF ball_x_motion(10) = '1' THEN
                ball_x_motion <= (NOT (ball_speed + hits(10 DOWNTO 0))) + 1;
            ELSIF ball_x_motion(10) = '0' THEN
                ball_x_motion <= (ball_speed + hits(10 DOWNTO 0));
            END IF;
            ball_y_motion <= (NOT (ball_speed + hits(10 DOWNTO 0))) + 1; -- set vspeed to (- ball_speed) pixels
    END IF;
    -- compute next ball vertical position
    -- variable temp adds one more bit to calculation to fix unsigned underflow problems
    -- when ball_y is close to zero and ball_y_motion is negative
    temp := ('0' & ball_y) + (ball_y_motion(10) & ball_y_motion);
    IF game_on = '0' THEN
        ball_y <= CONV_STD_LOGIC_VECTOR(440, 11);
```

```
        ELSIF temp(11) = '1' THEN
            ball_y <= (OTHERS => '0');
        ELSE ball_y <= temp(10 DOWNTO 0); -- 9 downto 0
        END IF;
        -- compute next ball horizontal position
        -- variable temp adds one more bit to calculation to fix unsigned underflow problems
        -- when ball_x is close to zero and ball_x_motion is negative
        temp := ('0' & ball_x) + (ball_x_motion(10) & ball_x_motion);
        IF temp(11) = '1' THEN
            ball_x <= (OTHERS => '0');
        ELSE ball_x <= temp(10 DOWNTO 0);
        END IF;
    END PROCESS;
END Behavioral;
```

In order to count the number of times the ball has been hit, a new signal named hits was created. The hits variable is incremented in the mball process whenever the ball bounces off the bat. An issue was encountered where hits was being incremented multiple times per hit. This was solved by creating the hit_on signal. This signal is set to ‘1’ each time the ball bounces off the top of the screen and ‘0’ when hits is incremented. An if statement was then added to only increment hits when hit_on = ‘1’. This prevents any additional hits from being registered until the ball has bounced off the top and returned to the bat. Hits is reset to 0 at the start of a new serve. A new output was added to the entity called score. Hits is passed to score for use in pong.vhd. The only change made in pong.vhd was the addition of score to the port mapping. Score was mapped to display which displayed the value of hits on the 7-segment display. In order to make the bat shrink with each hit all references to the bat's width were subtracted from by the number of hits. This was done by replacing bat_w with (bat_w - CONV_INTEGER(hits)). A similar method was used to make the ball speed up. All references to ball_speed were replaced with (ball_speed + hits(10 DOWNTO 0).