# Lab 3

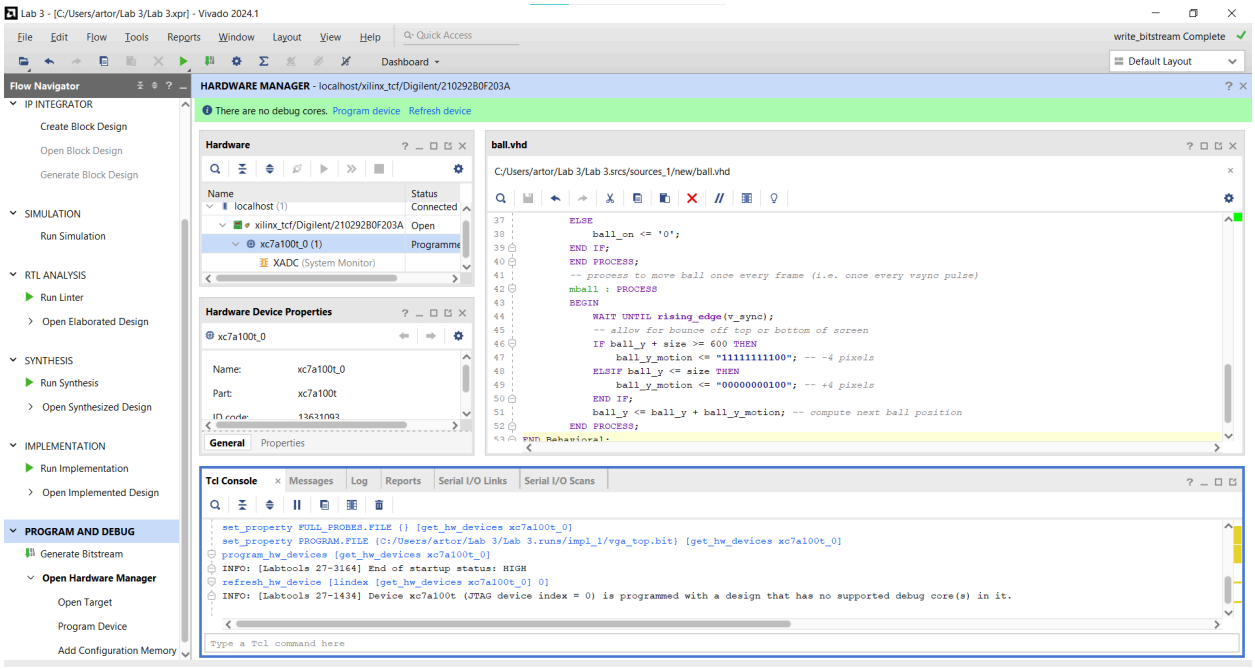
Adrian Torres and Angel Ordonez Retamar

**Part 1**

Original Output:

https://youtu.be/YKpVQNdnOCE

Code uploaded to board:




**Part 2**

Modified Output:

https://youtu.be/bA0wX2RmdpU

Explanation of Code Changes

Originally we had set the bounds of x to be the same as they were for y so the ball bounced in a square that did not take up the full width of the monitor. We then correctly set the bounds of x to be 800 which made the ball bounce around the full screen.

Modified Code “ball.vhd”:

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

ENTITY 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);
                red       : OUT STD_LOGIC;
```

```
                green     : OUT STD_LOGIC;
                blue      : OUT STD_LOGIC
        );
END ball;

ARCHITECTURE Behavioral OF ball IS
        CONSTANT size  : INTEGER := 32;
        SIGNAL ball_on : STD_LOGIC; -- indicates whether ball is over current pixel position
        -- 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);
        -- current ball motion - initialized to +4 pixels/frame
        SIGNAL ball_y_motion : STD_LOGIC_VECTOR(10 DOWNTO 0) := "00000000100";
        SIGNAL ball_x_motion : STD_LOGIC_VECTOR(10 DOWNTO 0) := "00000000100";
BEGIN
        red <= '1'; -- color setup for red ball on white background
        green <= NOT ball_on;
        blue  <= '1';
        -- process to draw ball current pixel address is covered by ball position


        bdraw : PROCESS (ball_x, ball_y, pixel_row, pixel_col) IS
   BEGIN
           IF (((conv_integer(pixel_col) - conv_integer(ball_x)) * (conv_integer(pixel_col) -
conv_integer(ball_x))) + ((conv_integer(pixel_row) - conv_integer(ball_y)) *
(conv_integer(pixel_row) - conv_integer(ball_y))) <= (size * size)) THEN
                ball_on <= '1';
           ELSE
                ball_on <= '0';
        END IF;
END PROCESS;


                -- process to move ball once every frame (i.e. once every vsync pulse)
                mball : PROCESS
                BEGIN
                        WAIT UNTIL rising_edge(v_sync);
                        -- allow for bounce off top or bottom of screen
                        IF ball_y + size >= 600 THEN
                                ball_y_motion <= "11111111100"; -- -4 pixels
                        ELSIF ball_y <= size THEN
                                ball_y_motion <= "00000000100"; -- +4 pixels
                        END IF;
                        IF ball_x + size >= 800 THEN
                                ball_x_motion <= "11111111100"; -- -4 pixels
                        ELSIF ball_x <= size THEN
                                ball_x_motion <= "00000000100"; -- +4 pixels
                        END IF;
                        ball_y <= ball_y + ball_y_motion; -- compute next ball position
                        ball_x <= ball_x + ball_x_motion;
                END PROCESS;
```

```
END Behavioral;
```

Original Code “ball.vhd”:

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

ENTITY 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);
		red       : OUT STD_LOGIC;
		green     : OUT STD_LOGIC;
		blue      : OUT STD_LOGIC
	);
END ball;

ARCHITECTURE Behavioral OF ball IS
	CONSTANT size  : INTEGER := 8;
	SIGNAL ball_on : STD_LOGIC; -- indicates whether ball is over current pixel position
	-- 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);
	-- current ball motion - initialized to +4 pixels/frame
	SIGNAL ball_y_motion : STD_LOGIC_VECTOR(10 DOWNTO 0) := "00000000100";
BEGIN
	red <= '1'; -- color setup for red ball on white background
	green <= NOT ball_on;
	blue  <= NOT ball_on;
	-- process to draw ball current pixel address is covered by ball position
	bdraw : PROCESS (ball_x, ball_y, pixel_row, pixel_col) IS
	BEGIN
		IF (pixel_col >= ball_x - size) AND
		 (pixel_col <= ball_x + size) AND
			(pixel_row >= ball_y - size) AND
			(pixel_row <= ball_y + size) THEN
				ball_on <= '1';
		ELSE
			ball_on <= '0';
		END IF;
		END PROCESS;
		-- process to move ball once every frame (i.e. once every vsync pulse)
		mball : PROCESS
		BEGIN
			WAIT UNTIL rising_edge(v_sync);
			-- allow for bounce off top or bottom of screen
			IF ball_y + size >= 600 THEN
				ball_y_motion <= "11111111100"; -- -4 pixels
			ELSIF ball_y <= size THEN
```

```
                    ball_y_motion <= "00000000100"; -- +4 pixels
                END IF;
                ball_y <= ball_y + ball_y_motion; -- compute next ball position
            END PROCESS;
    END Behavioral;
```

The modifications:

For the code I did 4 main modifications and I highlighted them in different colors to make it easier to discern. The red highlighted code adjusts the object's size by increasing its numerical value. In the green highlighted code, I modified the object's color by setting the blue component to ‘1,’ resulting in a purple color by having both red and blue set to on. The yellow highlighted code introduces x direction motion by copying the y-motion code, with the x size set to 800 instead of 600. Finally, the blue highlighted code changes the object’s shape from a square to a circle by using the circle equation with the given constants.