

```

1 -----
2 -- Engineer: Zachary Montoya
3 -- Submitted Date: 12-15-22
4 -- Module Name:    Top - Behavioral
5 -- Project Name:   Asteroids
6 -- Target Devices: Zybo Z7-10
7 -- Tool versions:  Vivado 2020.2
8 --
9 -- Comment:
10 --      This file was entirely created the engineer listed above.
11 --
12 -----
13
14 --
=====
=====
15 --
=====
=====
16 library ieee;
17 use ieee.std_logic_1164.all;
18 use ieee.numeric_std.all;
19
20 entity missile is
21     generic(
22         missile_velocity: integer := 1
23     );
24
25     port(
26         clk, reset: in std_logic;
27         btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
28         video_on: in std_logic;
29         pixel_x, pixel_y: in std_logic_vector(9 downto 0);
30         missile_on_out: out std_logic;
31         missile_rgb_out: out std_logic_vector(2 downto 0);
32         rom_selector_out: in std_logic_vector(2 downto 0);
33         bar_x_reg_out, bar_y_reg_out: in unsigned( 9 downto 0);
34         fire_out, fire_btn_signal_out: in std_logic;
35         fire_ready_out: out std_logic;
36         missile_left_out, missile_right_out, missile_top_out, missile_bottom_out: out
            unsigned(9 downto 0)
37     );
38 end missile;
39
40 --
=====
=====
42 --
=====
=====
43
44 architecture rtl of missile is
45
46     -- Signal used to control speed of ball and how often pushbuttons are checked for
    paddle movement.
47     signal refr_tick: std_logic;
48
49     -- x, y coordinates (0,0 to (639, 479)
50     signal pix_x, pix_y: unsigned(9 downto 0);
51
52     -- WALL1 - LEFT
53     constant WALL1_X_L: integer := 0;
54     constant WALL1_X_R: integer := 20;
55
56     -- WALL2 - RIGHT
57     constant WALL2_X_L: integer := 619;
58     constant WALL2_X_R: integer := 639;
59
60     -- WALL3 - BOTTOM
61     constant WALL3_X_T: integer := 409;
62     constant WALL3_X_B: integer := 479;
63
64     -- WALL4 - TOP
65     constant WALL4_X_T: integer := 0;
66     constant WALL4_X_B: integer := 20;
67

```

```

68 -- Square ball -- ball left, right, top and bottom all vary. Left and top driven by
   registers below.
69     constant BALL_SIZE: integer := 16; -----CHANGED Increased to 16
70     signal ball_x_l, ball_x_r: unsigned(9 downto 0);
71     signal ball_y_t, ball_y_b: unsigned(9 downto 0);
72
73 -- Reg to track left and top boundary
74     signal ball_x_reg, ball_x_next, ball_x_next_mux: unsigned(9 downto 0);
75     signal ball_y_reg, ball_y_next, ball_y_next_mux: unsigned(9 downto 0);
76
77 -- reg to track ball speed
78     signal x_delta_reg, x_delta_next: unsigned(9 downto 0);
79     signal y_delta_reg, y_delta_next: unsigned(9 downto 0);
80
81 -- ball movement can be pos or neg
82     constant BALL_V_P: unsigned(9 downto 0) := to_unsigned(missile_velocity,10);
83     constant BALL_V_N: unsigned(9 downto 0) := unsigned(to_signed(-
missile_velocity,10));
84     constant BALL_V_NULL: unsigned(9 downto 0) := to_unsigned(0,10);
85
86 -- round ball image
87     type rom_type is array(0 to 15) of std_logic_vector(0 to 15); -----Changed
   from array(0 to 7)
88     constant BALL_ROM_UP: rom_type := (
89         "0000000110000000",
90         "1100000110000011",
91         "1100000000000011",
92         "0000000000000000",
93         "0000000000000000",
94         "0000000110000000",
95         "1100000110000011",
96         "1100000110000011",
97         "1100000110000011",
98         "1100000110000011",
99         "1100000110000011",
100        "1100000000000011",
101        "0000000000000000",
102        "0000000000000000",
103        "0000000000000000",
104        "0000000000000000");
105     constant BALL_ROM_DOWN: rom_type := (
106        "0000000000000000",
107        "0000000000000000",
108        "0000000000000000",
109        "0000000000000000",
110        "1100000110000011",
111        "1100000000000011",
112        "1100000000000011",
113        "1100000000000011",
114        "1100000000000011",
115        "1100000000000011",
116        "0000000110000000",
117        "0000000000000000",
118        "0000000000000000",
119        "1100000000000011",
120        "1100000110000011",
121        "0000000110000000");
122     constant BALL_ROM_LEFT: rom_type := (
123        "0110001111110000",
124        "0110001111110000",
125        "0000000000000000",
126        "0000000000000000",
127        "0000000000000000",
128        "0000000000000000",
129        "0000000000000000",
130        "1100011111000000",
131        "1100011111000000",
132        "0000000000000000",
133        "0000000000000000",
134        "0000000000000000",
135        "0000000000000000",
136        "0000000000000000",
137        "0110001111110000",
138        "0110001111110000");
139     constant BALL_ROM_RIGHT: rom_type := (
140        "000011111100110",
141        "000011111100110",
142        "0000000000000000",
143        "0000000000000000",

```

```

144     "0000000000000000",
145     "0000000000000000",
146     "0000000000000000",
147     "000001111100011",
148     "000001111100011",
149     "0000000000000000",
150     "0000000000000000",
151     "0000000000000000",
152     "0000000000000000",
153     "0000000000000000",
154     "000011111000110",
155     "000011111000110");
156 -- Testing
157 -- 0 is to paint the background and 1 will be to use the ball
158 -- Can make more than one bit to make characters and other sprites.
159
160
161 -- ball is 8x8, the address only needs to be 3 bits then.
162 -- data will first be read as a row
163 -- rom_bit will go to the bit in the row
164 signal rom_addr, rom_col: unsigned(3 downto 0); -----Changed
to 4 bits from unsigned (2 downto 0)
165 signal rom_data: std_logic_vector(15 downto 0); -----Changed
from (7 downto 0)
166 signal rom_bit: std_logic;
167
168 -- object output signals -- new signal to indicate if scan coord is within ball
169 signal sq_ball_on, rd_ball_on: std_logic;
170 signal ball_rgb: std_logic_vector(2 downto 0);
171 signal fire_ready_reg, fire_ready_next: std_logic;
172 signal rom_selector_reg, rom_selector_next: std_logic_vector(2 downto 0);
173
174
175 --
=====
176 begin
177
178 missile_on_out <= rd_ball_on;
179 missile_rgb_out <= ball_rgb;
180 fire_ready_out <= fire_ready_next;
181
182
183 with fire_out select
184     ball_x_next_mux <= ball_x_reg_out when '0',
185                       ball_x_next when '1';
186
187 with fire_out select
188     ball_y_next_mux <= ball_y_reg_out when '0',
189                       ball_y_next when '1';
190
191 process (clk, reset)
192     begin
193         if (reset = '1') then
194             ball_x_reg <= ball_x_reg_out;
195             ball_y_reg <= ball_y_reg_out;
196             x_delta_reg <= ("000000000");
197             y_delta_reg <= ("000000000");
198             fire_ready_reg <= '0';
199             rom_selector_reg <= "111";
200         elsif (clk'event and clk = '1') then
201             ball_x_reg <= ball_x_next_mux;
202             ball_y_reg <= ball_y_next_mux;
203             x_delta_reg <= x_delta_next;
204             y_delta_reg <= y_delta_next;
205             fire_ready_reg <= fire_ready_next;
206             rom_selector_reg <= rom_selector_next;
207         end if;
208     end process;
209
210 --
=====
211     pix_x <= unsigned(pixel_x);
212     pix_y <= unsigned(pixel_y);
213
214 -- Refr_tick: 1-clock tick asserted at start of v_sync, e.g., when the screen is
refreshed -- speed is 60 Hz
215     -- refr_tick <= '1' when (pix_y = 1) and (pix_x = 1) else '0';

```

```

216     refr_tick <= '1' when (((pix_y = 1) or (pix_y = 73) or (pix_y = 146) or (pix_y =
217 220) or (pix_y = 293) or (pix_y = 365)) and (pix_x = 1)) else '0';
217 -- refr_tick <= '1' when (pix_y = 1) or (pix_y = 240) else '0';
218 --
=====
=====
219
220     process(rom_selector_reg, rom_selector_out)
221     begin
222         rom_selector_next <= rom_selector_reg;
223         if (btn(0) = '1' and fire_btn_signal_out = '0') then --Add
fire_button_ready_signal!!
224             rom_selector_next <= rom_selector_out;
225         end if;
226     end process;
227
228 --
=====
=====
229 -- set coordinates of square ball.
230     ball_x_l <= ball_x_reg;
231     ball_y_t <= ball_y_reg;
232     ball_x_r <= ball_x_l + BALL_SIZE - 1;
233     ball_y_b <= ball_y_t + BALL_SIZE - 1;
234
235 -- pixel within square ball
236     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';
237
238 -- Map scan coord to ROM addr/col -- use low order three bits of pixel and ball
positions. ROM row
239     rom_addr <= pix_y(3 downto 0) - ball_y_t(3 downto 0); ----- CHANGED
TO 4 BITS
240
241 -- ROM column
242     rom_col <= pix_x(3 downto 0) - ball_x_l(3 downto 0);----- CHANGED
TO 4 BITS
243
244 -- Get row data
245     process(fire_out,rom_selector_out)
246     begin
247         if (fire_out = '0') then
248             if (rom_selector_out = "110") then
249                 rom_data <= BALL_ROM_UP(to_integer(rom_addr));
250             elsif (rom_selector_out = "000") then
251                 rom_data <= BALL_ROM_DOWN(to_integer(rom_addr));
252             elsif (rom_selector_out = "100") then
253                 rom_data <= BALL_ROM_LEFT(to_integer(rom_addr));
254             elsif (rom_selector_out = "010") then
255                 rom_data <= BALL_ROM_RIGHT(to_integer(rom_addr));
256             end if;
257         elsif (fire_out = '1') then
258             if (rom_selector_reg = "110") then
259                 rom_data <= BALL_ROM_UP(to_integer(rom_addr));
260             elsif (rom_selector_reg = "000") then
261                 rom_data <= BALL_ROM_DOWN(to_integer(rom_addr));
262             elsif (rom_selector_reg = "100") then
263                 rom_data <= BALL_ROM_LEFT(to_integer(rom_addr));
264             elsif (rom_selector_reg = "010") then
265                 rom_data <= BALL_ROM_RIGHT(to_integer(rom_addr));
266             end if;
267         end if;
268     end process;
269
270
271 -- Get column bit
272     rom_bit <= rom_data(to_integer(rom_col));
273
274 -- Turn ball on only if within square and the ROM bit is 1.
275     rd_ball_on <= '1' when (sq_ball_on = '1') and (rom_bit = '1') else '0';
276     ball_rgb <= "111"; -- WHITE BALL COLOR
277
278 -- Update the ball position 60 times per second.
279     ball_x_next <= ball_x_reg + x_delta_reg when refr_tick = '1' else ball_x_reg;
280     ball_y_next <= ball_y_reg + y_delta_reg when refr_tick = '1' else ball_y_reg;
281
282
283 -- Set the value of the next ball position according to the boundaries.

```

```

284  -- 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)
285  -- begin
286  --     x_delta_next <= x_delta_reg; -- don't move
287  --     y_delta_next <= y_delta_reg; -- don't move
288  --     fire_ready_next <= fire_ready_reg;
289
290  --     if (rom_selector_reg = "110" and ball_y_t >= WALL4_X_B and ball_y_t < 479 )
then -- DOWN
291  --         y_delta_next <= BALL_V_N;
292  --         x_delta_next <= BALL_V_NULL;
293  --     elsif (rom_selector_reg = "110" and ball_y_t = WALL4_X_B and ball_y_t < 479
) then -- DOWN
294  --         y_delta_next <= BALL_V_NULL;
295  --         x_delta_next <= BALL_V_NULL;
296  --         fire_ready_next <= '1';
297
298  --     elsif (rom_selector_reg = "000" and ball_y_b <= WALL3_X_T and ball_y_b > 0)
then -- UP
299  --         y_delta_next <= BALL_V_P;
300  --         x_delta_next <= BALL_V_NULL;
301  --     elsif (rom_selector_reg = "000" and ball_y_b = WALL3_X_T and ball_y_b > 0)
then -- UP
302  --         y_delta_next <= BALL_V_NULL;
303  --         x_delta_next <= BALL_V_NULL;
304  --         fire_ready_next <= '1';
305
306  --     elsif (rom_selector_reg = "010" and ball_x_l >= WALL1_X_R and ball_x_l > 0)
then -- RIGHT
307  --         x_delta_next <= BALL_V_N;
308  --         y_delta_next <= BALL_V_NULL;
309  --     elsif (rom_selector_reg = "010" and ball_x_l = WALL1_X_R and ball_x_l > 0)
then -- RIGHT
310  --         y_delta_next <= BALL_V_NULL;
311  --         x_delta_next <= BALL_V_NULL;
312  --         fire_ready_next <= '1';
313
314  --     elsif (rom_selector_reg = "100" and ball_x_r <= WALL2_X_L and ball_x_r <
639) then -- LEFT
315  --         x_delta_next <= BALL_V_P;
316  --         y_delta_next <= BALL_V_NULL;
317  --     elsif (rom_selector_reg = "100" and ball_x_r = WALL2_X_L and ball_x_r < 639)
then -- LEFT
318  --         y_delta_next <= BALL_V_NULL;
319  --         x_delta_next <= BALL_V_NULL;
320  --         fire_ready_next <= '1';
321  --     end if ;
322
323  -- Set the value of the next ball position according to the boundaries.
324  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)
325  begin
326  x_delta_next <= x_delta_reg; -- don't move
327  y_delta_next <= y_delta_reg; -- don't move
328  fire_ready_next <= fire_ready_reg;
329
330  if (rom_selector_reg = "110" and ball_y_t >= WALL4_X_B and ball_y_t < 479 )
then -- DOWN
331  y_delta_next <= BALL_V_N;
332  x_delta_next <= BALL_V_NULL;
333
334  if ( ball_y_t <= WALL4_X_B ) then
335  y_delta_next <= BALL_V_NULL;
336  x_delta_next <= BALL_V_NULL;
337  fire_ready_next <= '1';
338  else
339  fire_ready_next <= '0';
340  end if;
341
342  elsif (rom_selector_reg = "000" and ball_y_b <= WALL3_X_T and ball_y_b > 0)
then -- UP
343  y_delta_next <= BALL_V_P;
344  x_delta_next <= BALL_V_NULL;
345
346  if (ball_y_b >= WALL3_X_T) then
347  y_delta_next <= BALL_V_NULL;
348  x_delta_next <= BALL_V_NULL;
349  fire_ready_next <= '1';
350  else

```

```

351         fire_ready_next <= '0';
352     end if;
353
354     elsif (rom_selector_reg = "010" and ball_x_l >= WALL1_X_R and ball_x_l > 0)
355 then -- RIGHT
356     x_delta_next <= BALL_V_N;
357     y_delta_next <= BALL_V_NULL;
358
359     if (ball_x_l <= WALL1_X_R ) then
360         y_delta_next <= BALL_V_NULL;
361         x_delta_next <= BALL_V_NULL;
362         fire_ready_next <= '1';
363     else
364         fire_ready_next <= '0';
365     end if;
366
367     elsif (rom_selector_reg = "100" and ball_x_r <= WALL2_X_L and ball_x_r < 639)
368 then -- LEFT
369     x_delta_next <= BALL_V_P;
370     y_delta_next <= BALL_V_NULL;
371
372     if (ball_x_r >= WALL2_X_L) then
373         y_delta_next <= BALL_V_NULL;
374         x_delta_next <= BALL_V_NULL;
375         fire_ready_next <= '1';
376     else
377         fire_ready_next <= '0';
378     end if;
379 end if ;
380
381 end process;
382
383
384 end rtl;
385
386
387
388 -- - - =====

```