



# Final Project Report

## FPGA Videogame

## Intermediate Logic Design

Professor: Jim, Plusquellic

Student: Montoya, Zachary

Submitted: December 15<sup>th</sup>, 2022

Albuquerque, NM



## Summary

This report is to summarize the work carried out for the final project for ECE338 Intermediate Logic Design, a video game implemented on the Xilinx Zynq Z7-10A SoC written in VHDL. A revision yielding strong elements of the games vision was achieved that features a vibrant border, multiple module instantiated asteroids, a realistic space cruiser that is capable of 2-dimensional travel through outer space that can fire a realistic missile cluster! A revision history was included within the Weekly Update Tracker toward the end of this report to capture the progress of this project.



## Discussion

This project truly was a wonderful learning experience with VHDL and I whole heartly agree with the remarks given that most of the learning occurs within the last three weeks of the semester where the focus is on this. I was able to tune my iteration speed on this project by being generating revisions to the VHDL faster using a windows computer versus my OSX machine and I felt this greatly enhanced my learning experience.

The largest hurdle was to separate the asteroids and paddle into different modules, I latter found that the issue resided from a problem I accidentally created with my HDMI driver when moving the project around ironically. After this I was able to generate the missiles which are quite interesting because they update state with the ship, including velocity and ROM orientation. In addition, the missiles also capture the orientation of the ship during firing. This was an interesting task because there were several iterations where the missiles would either change orientation after fired when the ship changed orientation or would relaunch into a different direction when the ship changed orientation and fire button was pressed. These bugs were resolved using the classical ready and start signals between the spaceship and the missile, the missile would only issue ready once it has reached the boundary of the screen only allowing for one shot to be fired at a time.

Per recommendation a text module was beginning to be implemented to capture a score along with a counter of when the missiles contact the asteroids or when the asteroids contact the ship. The core of this interaction would also have been a collider which was also not implemented, albeit nonfunctioning attempts were made to track the boundary positions of the entities using ports.

## Conclusion

Ultimately, in the same breath I personally recognize this project implementation as successful and am proud of the work created especially given the time and resources, but also wish there was more time to perfect the functionality of this game.



## Weekly Update Tracker

The purpose of this section is to provide regular updates, occurring every Wednesday, describing what has been worked on and what features were added to the game. Because this report is cumulative, the current updates will be added on top of the previous week's updates to form a reverse chronological order the features and revisions made. The authors responsible for the revisions created where included in the respective revision number. See Table 1 below.

Table 1 - Weekly Update Tracker

<b>Date: 12-07-22   Revision Number: 4</b>
<b>Title: Multiple Asteroid Instantiations via modules, missiles, ship ROM, and the beginnings of text</b>
<b>Authors: Zachary Montoya</b>
The objective of this revision was to finalize the game including all the options committed to during the initial project proposal. This however was not possible given the time and resource constraints, as this group project migrated into a solo effort. I am however incredibly proud to provide an update that the scope of this final revision does include multiple asteroid instantiations via modules, missiles, a newly improved orientation dependent ship ROM, and the foundation for a text score. Given that these updates are too large to include within snippets of figures it is recommended that the reviewer of this document reference the attachments include with this document.

**Date: 12-07-22**
**Revision Number: 3**
**Title: Walls, 2D Ship Movement, multiple asteroids**
**Authors: Zachary Montoya**

The objective of this revision is to add (1) walls for the game to interact within, (2) 2D dimensional ship movement that has independent button control, and (3) multiple asteroids. However multiple asteroids were only able to be implemented within one file not multiple modules that are instantiated via a master controller this is still in progress.

**Figure 1 – Additional game boundaries**

```
-- WALL1 - LEFT
constant WALL1_X_L: integer := 0;
constant WALL1_X_R: integer := 10;
-- WALL2 - RIGHT
constant WALL2_X_L: integer := 629;
constant WALL2_X_R: integer := 639;
-- WALL3 - BOTTOM
constant WALL3_X_T: integer := 469;
constant WALL3_X_B: integer := 479;
-- WALL4 - TOP
constant WALL4_X_T: integer := 0;
constant WALL4_X_B: integer := 10;
```

**Figure 2 – Modified constraints to accommodate the new btns**

```
12 ##Switches
13 set_property -dict { PACKAGE_PIN G15 IOSTANDARD LVCMS33 } [get_ports { reset }]; #IO_L19N_T3_VREF_35 Sch=sw[0]
14 #set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMS33 } [get_ports { sw_g }]; #IO_L24P_T3_34 Sch=sw[1]
15 #set_property -dict { PACKAGE_PIN W13 IOSTANDARD LVCMS33 } [get_ports { sw_b }]; #IO_L4N_T0_34 Sch=sw[2]
16 #set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=sw[3]
17
18
19 ##Buttons
20 set_property -dict { PACKAGE_PIN K18 IOSTANDARD LVCMS33 } [get_ports { btn[0] }]; #IO_L12N_T1_MRCC_35 Sch=btn[0]
21 set_property -dict { PACKAGE_PIN P16 IOSTANDARD LVCMS33 } [get_ports { btn[1] }]; #IO_L24N_T3_34 Sch=btn[1]
22 set_property -dict { PACKAGE_PIN K19 IOSTANDARD LVCMS33 } [get_ports { btn[2] }]; #IO_L10P_T1_AD11P_35 Sch=btn[2]
23 set_property -dict { PACKAGE_PIN Y16 IOSTANDARD LVCMS33 } [get_ports { btn[3] }]; #IO_L7P_T1_34 Sch=btn[3]
```

**Figure 3 – Additional game boundaries**

```
-- -----
-- Process bar movement requests
process( bar_y_reg, bar_y_b, bar_y_t,bar_x_reg,bar_x_L,bar_x_R, refr_tick, btn)
begin
    bar_y_next <= bar_y_reg; -- no move
    bar_x_next <= bar_x_reg; -- no move--R2

    if ( refr_tick = '1' ) then

    -- if btn 1 pressed and paddle not at bottom yet
        if ( btn(0) = '1' and bar_y_b < (MAX_Y - 1 - BAR_V) ) then
            bar_y_next <= bar_y_reg + BAR_V; -- move down

    -- if btn 1 NOT pressed and bar not at top yet
        elsif ( btn(1) = '1' and bar_y_t > BAR_V ) then
            bar_y_next <= bar_y_reg - BAR_V; -- move up
```

Date: 11-30-22 | Revision Number: 2

## Title: Asteroid Element Size Increase and Horizontal Logic of the Paddle

**Authors: Anindya Bal and Zachary Montoya**

The objective for this revision was to: (1) increase the dimensions of previously modified asteroid element, (2) improve the asteroid element's visual design, (3) include horizontal movement to the paddle. This third objective is motivated by a larger goal of turning the paddle into a spaceship used for destroying asteroids! These three minor objectives were met. See images below of some of the core code changes to implement these objectives:

Figure 4 - Lines 54 of pong\_graph\_st.vhd file

```
Diagram x Address Editor x Address Map x pong_graph_st.vhd x ZYBO_Z7-10_main.xsf x
/home/zacharymontoya/Desktop/Labs/AlienControlledAstroids2/AlienControlledAstroids/AlienControlledAstroids.srsc/sourc
Q | < | > | X | // | { | } | ? | 52
53 -- Square ball -- ball left, right, top and bottom all vary. Left and top driven by registers below.
54 constant BALL_SIZE: integer := 16; -----CHANGED Increased to 16
55 signal ball_x_l, ball_x_r: unsigned(9 downto 0);
56 signal ball_y_t, ball_y_b: unsigned(9 downto 0);
```

Figure 5 – 16x16 BALL ROM in the pong\_graph\_st.vhd file

```
Diagram x Address Editor x Address Map x pong_graph_st.vhd x
/home/zacharymontoya/Desktop/Labs/AlienControlledAstroids2/AlienControlledAstroids/A

Q H ← → X D // E ? | 67 constant BALL_V_P: unsigned(9 downto 0) := to_unsigned(2,10);
| 68 constant BALL_V_N: unsigned(9 downto 0) := unsigned(to_signed(-2,10));
| 69
| 70 -- round ball image
| 71 type rom_type is array(0 to 15) of std_logic_vector(0 to 15); -----
| 72 constant BALL_ROM: rom_type := (
| 73     "000111111110000",
| 74     "001011001101100",
| 75     "011110110110110",
| 76     "111101111011110",
| 77     "101101101101111",
| 78     "111110111101011",
| 79     "111011011011111",
| 80     "110111110011111",
| 81     "110111111101111",
| 82     "111111111111111",
| 83     "101110111111111",
| 84     "111011001111111",
| 85     "011111111111110",
| 86     "001011110110110",
| 87     "000110111111100",
| 88     "000011111111000");
| 89
```

**Date: 11-30-22 | Revision Number: 2**

**Title: Asteroid Element Size Increase and Horizontal Logic of the Paddle**

**Authors: Anindya Bal and Zachary Montoya**

Figure 6 – Paddle Movement Requests in the pong\_graph\_st.vhd file

```

Diagram  x Address Editor  x Address Map  x pong_graph_st.vhd *  x ZYBO_Z7-
/home/zacharymontoya/Desktop/Labs/AlienControlledAstroids2/AlienControlledAstroids/AlienController
Q | H | ← | → | X | C | D | X | // | ■ | Q |
151 -- Process bar movement requests
152   process( bar_y_reg, bar_y_b, bar_y_t, bar_x_reg, bar_x_L, bar_x_R, refr_tick, btn)
153     begin
154       bar_y_next <= bar_y_reg; -- no move
155       bar_x_next <= bar_x_reg; -- no move--R2
156
157     if ( refr_tick = '1' ) then
158
159     -- if btn 1 pressed and paddle not at bottom yet
160     if ( btn(1) = '1' and bar_y_b < (MAX_Y - 1 - BAR_V) ) then
161       bar_y_next <= bar_y_reg + BAR_V; -- move down
162
163     -- if btn 1 NOT pressed and bar not at top yet
164     elsif ( btn(1) = '0' and bar_y_t > BAR_V ) then
165       bar_y_next <= bar_y_reg - BAR_V; -- move up
166
167     -- if btn 0 pressed and bar not at RIGHT yet --R2
168     elsif ( btn(0) = '1' and bar_x_R < (MAX_X - 1 - BAR_V) ) then
169       bar_x_next <= bar_x_reg + BAR_V; -- move RIGHT--R2
170     -- if btn 0 NOT pressed and bar not at RIGHT yet
171     elsif ( btn(0) = '0' and bar_x_L > BAR_V ) then
172       bar_x_next <= bar_x_reg - BAR_V; --move LEFT
173
174   end if;
175 end process;

```

Date: 11-23-22	Revision Number: 1
<b>Title: ROM Mask Manipulation</b>	
<b>Authors: Zachary Montoya</b>	

The objective for this revision was to modify the shape of the ROM mask to change the shape of the outputted image. This is usefully to provide a variety of different shapes for the BALL element to display during the code execution. Note that the ROM mask map and associated BALL element was left at its original 8x8 dimensions, and the mask map was changed so the ball would visibly appear as an asteroid.

Figure 7 – 8x8 BALL ROM in the pong\_graph\_st.vhd file

```
65  -- round ball image
66  type rom_type is array(0 to 7) of std_logic_vector(0 to 7);
67  constant BALL_ROM: rom_type:= (
68      "00100000",
69      "01110000",
70      "01111000",
71      "11111000",
72      "11111100",
73      "01111111",
74      "00111100",
75      "00111000");
76
```



Date: 11-21-22

Subject: ECE595/ECE338 Intermediate Logic Design Project Request

Dear Professor Plusquellic,

The purpose of this memorandum is to submit a formal request for the ECE595/ECE338 Intermediate Logic Design Project Description for your consideration. The video game implemented in VHDL selected for submission is the default game: ALIEN CONTROLLED ASTEROIDS with ALL options selected.

The options our team is seeking to include in our game's implementation is as follows:

1. Optional: Firing operations may also involve firing 'cluster bombs' that destroy all asteroids within a fixed radius (you should limit how often a user to use cluster bombs).
2. Optional: An alien spaceship can be added that is under control of a second player
3. Optional: The number of asteroids can increase in number and speed as the game progresses
4. Optional: Asteroids can gravitate automatically toward the space ship
5. Optional: Asteroids can be directed by an alien, which is controlled by a second player
6. Optional: You can add additional space ships after the user destroys a fixed number of asteroids.

We are confident that although this is not a novel game topic that a successful implementation shall yield full marks for the project assignment.

Kind regards,

Anindya Bal

Zachary Montoya

```

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 library work;
21 use work.DataTypes_pkg.all;
22
23 entity Top is
24     port (
25         Clk: in std_logic;
26         reset : in STD_LOGIC;
27         GPIO_Ins: in std_logic_vector(31 downto 0);
28         GPIO_Outs: out std_logic_vector(31 downto 0);
29         PNL_BRAM_addr: out std_logic_vector(12 downto 0);
30         PNL_BRAM_din: out std_logic_vector(15 downto 0);
31         PNL_BRAM_dout: in std_logic_vector(15 downto 0);
32         PNL_BRAM_we: out std_logic_vector(0 to 0);
33         hdmi_red : out STD_LOGIC_VECTOR(7 downto 0);
34         hdmi_green : out STD_LOGIC_VECTOR(7 downto 0);
35         hdmi_blue : out STD_LOGIC_VECTOR(7 downto 0);
36         hdmi_hsync : out STD_LOGIC;
37         hdmi_vsync : out STD_LOGIC;
38         hdmi_enable : out STD_LOGIC;
39         btn: in std_logic_vector(3 downto 0); --R2 increasing the array from 2 to 3.
40         sw: in std_logic_vector(0 downto 0) --R2 increasing the array from 2 to 3.
41         -- DEBUG_IN: in std_logic;
42         -- DEBUG_OUT: out std_logic
43     );
44 end Top;
45
46 architecture beh of Top is
47
48 -- GPIO INPUT BIT ASSIGNMENTS
49 constant IN_CP_RESET: integer := 31;
50 constant IN_CP_START: integer := 30;
51 constant IN_CP_LM_ULM_LOAD_UNLOAD: integer := 26;
52 constant IN_CP_LM_ULM_DONE: integer := 25;
53 constant IN_CP_HANDSHAKE: integer := 24;
54
55 -- GPIO OUTPUT BIT ASSIGNMENTS
56 constant OUT_SM_READY: integer := 31;
57 constant OUT_SM_HANDSHAKE: integer := 28;
58
59 -- Signal declarations
60 signal RESET_final: std_logic;
61
62 signal pixel_x: unsigned(10 downto 0);
63 signal pixel_y: unsigned(9 downto 0);
64 signal pixel_x_std: std_logic_vector(9 downto 0);
65 signal pixel_y_std: std_logic_vector(9 downto 0);
66
67 signal LM_ULM_start, LM_ULM_ready: std_logic;
68 signal LM_ULM_stopped, LM_ULM_continue: std_logic;
69 signal LM_ULM_done: std_logic;
70 signal LM_ULM_base_address: std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
71 signal LM_ULM_upper_limit: std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
72 signal LM_ULM_load_unload: std_logic;
73
74 signal DataIn: std_logic_vector(WORD_SIZE_NB-1 downto 0);

```

```

75 signal DataOut: std_logic_vector(WORD_SIZE_NB-1 downto 0);
76
77 signal graph_rgb: std_logic_vector(2 downto 0);
78 signal hdmi_enable_out: STD_LOGIC;
79
80 --NEW
81 signal wall1_on, wall2_on, wall3_on, wall4_on, bar_on, asteroid1_on,
asteroid2_on,missile_on: std_logic;
82 signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, bar_rgb, asteroid1_rgb,
asteroid2_rgb,missile_rgb,rgb_text: std_logic_vector(2 downto 0);
83 signal Left_Wall, Right_Wall, Bottom_Wall, Top_Wall: std_logic_vector(9 downto 0);
84 signal rom_selector: std_logic_vector(2 downto 0);
85 signal bar_x_reg, bar_y_reg: unsigned( 9 downto 0);
86 signal fire, fire_ready, fire_btn_signal: std_logic;
87
88 --
=====begin
89
90
91
92
93
94
95 -- Light up LED if LoadUnLoadMemMod is ready for a command
96 -- DEBUG_OUT <= LM_ULM_ready;
97
98 -- =====
99 -- INPUT control and status signals
100 -- Software (C code) plus hardware global reset
101 RESET_final <= GPIO_Ins(IN_CP_RESET) or reset;
102
103 -- Start signal from C program.
104 LM_ULM_start <= GPIO_Ins(IN_CP_START);
105
106 -- C program controls whether we are loading or unloading memory
107 LM_ULM_load_unload <= GPIO_Ins(IN_CP_LM_ULM_LOAD_UNLOAD);
108
109 -- C program asserts if done reading or writing memory (or a portion of it)
110 LM_ULM_done <= GPIO_Ins(IN_CP_LM_ULM_DONE);
111
112 -- Handshake signal
113 LM_ULM_continue <= GPIO_Ins(IN_CP_HANDSHAKE);
114
115 -- Data from C program
116 DataIn <= GPIO_Ins(WORD_SIZE_NB-1 downto 0);
117
118 -- =====
119 -- OUTPUT control and status signals
120 -- Tell C program whether LoadUnLoadMemMod is ready
121 GPIO_Outs(OUT_SM_READY) <= LM_ULM_ready;
122
123 -- Handshake signals
124 GPIO_Outs(OUT_SM_HANDSHAKE) <= LM_ULM_stopped;
125
126 -- Data to C program
127 GPIO_Outs(WORD_SIZE_NB-1 downto 0) <= DataOut;
128
129 -- =====
130 -- Setup memory base and upper_limit
131 LM_ULM_base_address <= std_logic_vector(to_unsigned(PN_BRAM_BASE,
PNL_BRAM_ADDR_SIZE_NB));
132 LM_ULM_upper_limit <= std_logic_vector(to_unsigned(PNL_BRAM_NUM_WORDS_NB -1,
PNL_BRAM_ADDR_SIZE_NB));
133
134 -- Secure BRAM access control module
135 LoadUnLoadMemMod: entity work.LoadUnLoadMem(beh)
136     port map(Clk=>Clk, RESET=>RESET, start=>LM_ULM_start, ready=>LM_ULM_ready,
load_unload=>LM_ULM_load_unload, stopped=>LM_ULM_stopped,
137         continue=>LM_ULM_continue, done=>LM_ULM_done,
base_address=>LM_ULM_base_address, upper_limit=>LM_ULM_upper_limit,
138         CP_in_word=>DataIn, CP_out_word=>DataOut,
139         PNL_BRAM_addr=>PNL_BRAM_addr, PNL_BRAM_din=>PNL_BRAM_din,
PNL_BRAM_dout=>PNL_BRAM_dout, PNL_BRAM_we=>PNL_BRAM_we);
140
141     hdmi_sync_i: entity work.hdmi_sync(rtl)
142         port map (clk=>Clk, reset=>reset, hdmi_hsync=>hdmi_hsync,
hdmi_vsync=>hdmi_vsync, hdmi_enable=>hdmi_enable_out, pixel_x=>pixel_x,
pixel_y=>pixel_y);

```

```

143
144     pixel_x_std <= std_logic_vector(pixel_x(9 downto 0));
145     pixel_y_std <= std_logic_vector(pixel_y);
146
147 asteroid1: entity work.asteroid(rtl)
148 generic map(asteroid_velocity=>2)
149 port map (clk=>Clk,
150             reset=>reset,
151             video_on=>hdmi_enable_out,
152             pixel_x=>pixel_x_std,
153             pixel_y=>pixel_y_std,
154             asteroid_on_out=>asteroid1_on,
155             asteroid_rgb_out => asteroid1_rgb);
156
157 asteroid2: entity work.asteroid(rtl)
158 generic map(asteroid_velocity=>3)
159 port map (clk=>Clk,
160             reset=>reset,
161             video_on=>hdmi_enable_out,
162             pixel_x=>pixel_x_std,
163             pixel_y=>pixel_y_std,
164             asteroid_on_out=>asteroid2_on,
165             asteroid_rgb_out => asteroid2_rgb);
166
167 walls: entity work.walls(rtl)
168 port map (clk=>Clk,
169             reset=>reset,
170             video_on=>hdmi_enable_out,
171             pixel_x=>pixel_x_std,
172             pixel_y=>pixel_y_std,
173             wall1_on_out=>wall1_on,
174             wall2_on_out=>wall2_on,
175             wall3_on_out=>wall3_on,
176             wall4_on_out=>wall4_on,
177             wall1_rgb_out=>wall1_rgb,
178             wall2_rgb_out=>wall2_rgb,
179             wall3_rgb_out=>wall3_rgb,
180             wall4_rgb_out=>wall4_rgb,
181             Left_Wall_Out=>Left_Wall,
182             Right_Wall_Out=>Right_Wall,
183             Bottom_Wall_Out=>Bottom_Wall,
184             Top_Wall_Out=>Top_Wall);
185
186 ship_i: entity work.space_ship(rtl)
187 port map (clk=>Clk,
188             reset=>reset,
189             btn=>btn,
190             SW=>SW,
191             video_on=>hdmi_enable_out,
192             pixel_x=>pixel_x_std,
193             pixel_y=>pixel_y_std,
194             bar_on_out=>bar_on,
195             bar_rgb_out=>bar_rgb,
196             bar_x_reg_out => bar_x_reg,
197             bar_y_reg_out => bar_y_reg,
198             rom_selector_out => rom_selector,
199             fire_out => fire,
200             fire_ready_out => fire_ready,
201             fire_btn_signal_out => fire_btn_signal
202 );
203
204 missile: entity work.missile(rtl)
205 generic map(missile_velocity=>1)
206 port map (clk=>Clk,
207             reset=>reset,
208             btn=>btn,
209             video_on=>hdmi_enable_out,
210             pixel_x=>pixel_x_std,
211             pixel_y=>pixel_y_std,
212             missile_on_out=>missile_on, --Add to Graph
213             missile_rgb_out => missile_rgb, --Add to Graph
214             bar_x_reg_out => bar_x_reg,
215             bar_y_reg_out => bar_y_reg,
216             rom_selector_out => rom_selector,
217             fire_out => fire,
218             fire_ready_out => fire_ready,
219             fire_btn_signal_out => fire_btn_signal
220 );
221

```

```

222  text: entity work.font_test_gen(arch)
223    port map (clk=>Clk,
224      video_on=>hdmi_enable_out,
225      pixel_x=>pixel_x_std,
226      pixel_y=>pixel_y_std,
227      rgb_text_out=>rgb_text
228    );
229
230  --
231  =====
232  -- turn on the appropriate color depending on the current pixel position.
233  process (hdmi_enable_out, wall1_on, wall2_on, wall3_on, wall4_on, bar_on,
234  asteroid2_on, asteroid1_on, wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, bar_rgb,
235  asteroid2_rgb, asteroid1_rgb)
236  begin
237    if (hdmi_enable_out = '0') then
238      graph_rgb <= "110"; -- blank
239    else
240      if (wall1_on = '1') then
241        graph_rgb <= wall1_rgb;
242      elsif (wall2_on = '1') then
243        graph_rgb <= wall2_rgb;
244      elsif (wall3_on = '1') then
245        graph_rgb <= wall3_rgb;
246      elsif (wall4_on = '1') then
247        graph_rgb <= wall4_rgb;
248      elsif (bar_on = '1') then
249        graph_rgb <= bar_rgb;
250      elsif (asteroid2_on = '1') then
251        graph_rgb <= asteroid2_rgb;
252      elsif (asteroid1_on = '1') then
253        graph_rgb <= asteroid1_rgb;
254      elsif (missile_on = '1') then
255        graph_rgb <= missile_rgb;
256      -- elsif (missile_on = '1') then
257      --   graph_rgb <= rgb_text;
258      else
259        graph_rgb <= "000"; -- Black bkgnd
260        graph_rgb <= rgb_text;
261      end if;
262    end if;
263  end process;
264
265  --  hdmi_red <= std_logic_vector(resize(pixel_x, 8)) when sw_r = '1' else (others
266  --    => '0');
267  --  hdmi_green <= std_logic_vector(resize(pixel_y, 8)) when sw_g = '1' else (others
268  --    => '0');
269  hdmi_red <= "11111111" when graph_rgb(0) = '1' else (others => '0');
270  hdmi_green <= "11111111" when graph_rgb(1) = '1' else (others => '0');
271  hdmi_blue <= "11111111" when graph_rgb(2) = '1' else (others => '0');
272
273  hdmi_enable <= hdmi_enable_out;
274
275 end beh;
276
277
```

```

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 asteroid is
21     generic(
22         asteroid_velocity: integer := 5
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         asteroid_on_out: out std_logic;
31         asteroid_rgb_out: out std_logic_vector(2 downto 0)
32     );
33 end asteroid;
34
35 --
===== =====
36 --
===== =====
37
38 architecture rtl of asteroid is
39
40 -- Signal used to control speed of ball and how often pushbuttons are checked for
41 -- paddle movement.
42     signal refr_tick: std_logic;
43
44 -- x, y coordinates (0,0 to (639, 479)
45     signal pix_x, pix_y: unsigned(9 downto 0);
46
47 -- Screen dimensions
48     constant MAX_X: integer := 640;
49     constant MAX_Y: integer := 480;
50
51     -- WALL1 - LEFT
52     constant WALL1_X_L: integer := 0;
53     constant WALL1_X_R: integer := 20;
54
55     -- WALL2 - RIGHT
56     constant WALL2_X_L: integer := 619;
57     constant WALL2_X_R: integer := 639;
58
59     -- WALL3 - BOTTOM
60     constant WALL3_X_T: integer := 409;
61     constant WALL3_X_B: integer := 479;
62
63     -- WALL4 - TOP
64     constant WALL4_X_T: integer := 0;
65     constant WALL4_X_B: integer := 20;
66
67     -- Paddle left, right, top, bottom and height -- left & right are constant. Top &
68     -- bottom are signals to allow movement. bar_y_t driven by register below.
69     --constant BAR_X_L: integer := 600;--R2
70     --constant BAR_X_R: integer := 603;--R2

```

```

69  -- signal bar_x_L,bar_x_R: unsigned(9 downto 0); --R2
70  -- signal bar_y_t, bar_y_b: unsigned(9 downto 0);
71  -- constant BAR_Y_SIZE: integer := 72;
72  -- constant BAR_X_SIZE: integer := 8; --R2
73
74  -- -- Reg to track top boundary (x position is fixed)
75  --     signal bar_y_reg, bar_y_next: unsigned( 9 downto 0);
76
77  -- -- Reg to track right boundary --R2
78  --     signal bar_x_reg, bar_x_next: unsigned( 9 downto 0); --R2
79
80  -- Bar moving velocity when a button is pressed -- the amount the bar is moved.
81  -- constant BAR_V: integer:= 4;
82
83  -- Square ball -- ball left, right, top and bottom all vary. Left and top driven by
84  -- registers below.
85  constant BALL_SIZE: integer := 16; -----CHANGED Increased to 16
86  signal ball_x_l, ball_x_r: unsigned(9 downto 0);
87  signal ball_y_t, ball_y_b: unsigned(9 downto 0);
88
89  -- Reg to track left and top boundary
90  signal ball_x_reg, ball_x_next: unsigned(9 downto 0);
91  signal ball_y_reg, ball_y_next: unsigned(9 downto 0);
92
93  -- reg to track ball speed
94  signal x_delta_reg, x_delta_next: unsigned(9 downto 0);
95  signal y_delta_reg, y_delta_next: unsigned(9 downto 0);
96
97  -- ball movement can be pos or neg
98  constant BALL_V_P: unsigned(9 downto 0):= to_unsigned(asteroid_velocity,10);
99  constant BALL_V_N: unsigned(9 downto 0):= unsigned(to_signed(-
asteroid_velocity,10));
100
101 -- round ball image
102 type rom_type is array(0 to 15) of std_logic_vector(0 to 15); -----Changed
from array(0 to 7)
103 constant BALL_ROM: rom_type:= (
104    "0001111111110000",
105    "0010110011011000",
106    "0111101101110100",
107    "1111011110111110",
108    "1011011101111111",
109    "1111101111011011",
110    "1110110110111111",
111    "1101111001111111",
112    "1101111111011111",
113    "1111011111111111",
114    "1110110011111111",
115    "0111111111111110",
116    "0010111101101100",
117    "0001101111111100",
118    "0000111111111100");
119
120  -- Testing
121  -- 0 is to paint the background and 1 will be to use the ball
122  -- Can make more than one bit to make characters and other sprites.
123
124
125  -- ball is 8x8, the address only needs to be 3 bits then.
126  -- data will first be read as a row
127  -- rom_bit will go to the bit in the row
128  signal rom_addr, rom_col: unsigned(3 downto 0); -----Changed
to 4 bits from unsigned (2 downto 0)
129  signal rom_data: std_logic_vector(15 downto 0); -----Changed
from (7 downto 0)
130  signal rom_bit: std_logic;
131
132  -- object output signals -- new signal to indicate if scan coord is within ball
133  -- signal wall1_on, wall2_on, wall3_on, wall4_on, bar_on, sq_ball_on, rd_ball_on:
std_logic;
134  -- signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, bar_rgb, ball_rgb:
std_logic_vector(2 downto 0);
135  signal wall1_on, wall2_on, wall3_on, wall4_on, sq_ball_on, rd_ball_on: std_logic;
136  signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, ball_rgb: std_logic_vector(2
downto 0);
137
138 ==
=====
```

```

=====
139 begin
140
141 asteroid_on_out <= rd_ball_on;
142 asteroid_rgb_out <= ball_rgb;
143
144 process (clk, reset)
145 begin
146 if (reset = '1') then
147 -- bar_y_reg <= (others => '0');
148 -- bar_x_reg <= (others => '0');--R2
149 ball_x_reg <= (others => '0');
150 ball_y_reg <= (others => '0');
151 x_delta_reg <= ("0000000100");
152 y_delta_reg <= ("0000000100");
153 elsif (clk'event and clk = '1') then
154 -- bar_y_reg <= bar_y_next;
155 -- bar_x_reg <= bar_x_next;--R2
156 ball_x_reg <= ball_x_next;
157 ball_y_reg <= ball_y_next;
158 x_delta_reg <= x_delta_next;
159 y_delta_reg <= y_delta_next;
160 end if;
161 end process;
162
163 --
=====
164 pix_x <= unsigned(pixel_x);
165 pix_y <= unsigned(pixel_y);
166
167 -- Refr_tick: 1-clock tick asserted at start of v_sync, e.g., when the screen is
168 -- refreshed -- speed is 60 Hz
169 refr_tick <= '1' when (pix_y = 1) and (pix_x = 1) else '0';
170
171 --
=====
172 -- -- wall1 left vertical stripe
173 -- wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
174 -- convert pix_x to pix_y to make horizontal
175 -- wall1_rgb <= "011"; -- paddle colors blue
176 -- -- wall2 right vertical stripe
177 -- wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
178 -- wall2_rgb <= "011"; -- paddle colors blue
179 -- -- wall3 left vertical stripe
180 -- wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
181 -- convert pix_x to pix_y to make horizontal
182 -- wall3_rgb <= "011"; -- paddle colors blue
183 -- -- wall4 right vertical stripe
184 -- wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
185 -- wall4_rgb <= "011"; -- paddle colors blue
186
187 --
=====
188 -- -- pixel within paddle
189 -- bar_x_L <= bar_x_reg;--R2
190 -- bar_x_R <= bar_x_L + BAR_X_SIZE - 1;--R2
191 -- bar_y_t <= bar_y_reg;
192 -- bar_y_b <= bar_y_t + BAR_Y_SIZE - 1;
193 -- bar_on <= '1' when (BAR_X_L <= pix_x) and (pix_x <= BAR_X_R) and (bar_y_t <=
194 -- pix_y) and (pix_y <= bar_y_b) else '0';
195 -- bar_rgb <= "100"; -- Red color
196
197 --
=====
198 -- set coordinates of square ball.
199 ball_x_l <= ball_x_reg;
200 ball_y_t <= ball_y_reg;
201 ball_x_r <= ball_x_l + BALL_SIZE - 1;
202 ball_y_b <= ball_y_t + BALL_SIZE - 1;
203
204 -- pixel within square ball
205 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';

```

```

204 -- Map scan coord to ROM addr/col -- use low order three bits of pixel and ball
205   rom_addr <= pix_y(3 downto 0) - ball_y_t(3 downto 0); ----- CHANGED
206   TO 4 BITS
207 -- ROM column
208   rom_col <= pix_x(3 downto 0) - ball_x_l(3 downto 0);----- CHANGED
209   TO 4 BITS
210 -- Get row data
211   rom_data <= BALL_ROM(to_integer(rom_addr));
212
213 -- Get column bit
214   rom_bit <= rom_data(to_integer(rom_col));
215
216 -- Turn ball on only if within square and the ROM bit is 1.
217   rd_ball_on <= '1' when (sq_ball_on = '1') and (rom_bit = '1') else '0';
218   ball_rgb <= "111"; -- WHITE BALL COLOR
219
220 -- Update the ball position 60 times per second.
221   ball_x_next <= ball_x_reg + x_delta_reg when refr_tick = '1' else ball_x_reg;
222   ball_y_next <= ball_y_reg + y_delta_reg when refr_tick = '1' else ball_y_reg;
223
224 -- Set the value of the next ball position according to the boundaries.
225   -- process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_t,
226   ball_y_b, bar_y_t, bar_y_b)
227   process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_t,
228   ball_y_b)
229   begin
230     x_delta_next <= x_delta_reg;
231     y_delta_next <= y_delta_reg;
232
233 -- Ball reached top, make offset positive
234   if ( ball_y_t < WALL4_X_B ) then --MAKE WALL4
235     y_delta_next <= BALL_V_P;
236
237 -- Reached bottom, make negative
238   elsif (ball_y_b > (WALL3_X_T - 1)) then --MAKE WALL3
239     y_delta_next <= BALL_V_N;
240
241 -- Reach wall1, bounce back
242   elsif (ball_x_l <= WALL1_X_R ) then
243     x_delta_next <= BALL_V_P;
244
245 -- Reach wall2, bounce back
246   elsif (ball_x_r > WALL2_X_L) then
247     x_delta_next <= BALL_V_N;
248
249 -- -- Right corner of ball inside bar
250 --     elsif ((BAR_X_L <= ball_x_r) and (ball_x_r <= BAR_X_R)) then
251 --       if ((bar_y_t <= ball_y_b) and (ball_y_t <= bar_y_b)) then
252 --         x_delta_next <= BALL_V_N;
253 --       end if;
254   end if;
255
256 end process;
257
258
259 end rtl;
260
261
```

```

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 asteroid is
21     generic(
22         asteroid_velocity: integer := 5
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         asteroid_on_out: out std_logic;
31         asteroid_rgb_out: out std_logic_vector(2 downto 0)
32     );
33 end asteroid;
34
35 --
===== =====
36 --
===== =====
37
38 architecture rtl of asteroid is
39
40 -- Signal used to control speed of ball and how often pushbuttons are checked for
41 -- paddle movement.
42     signal refr_tick: std_logic;
43
44 -- x, y coordinates (0,0 to (639, 479)
45     signal pix_x, pix_y: unsigned(9 downto 0);
46
47 -- Screen dimensions
48     constant MAX_X: integer := 640;
49     constant MAX_Y: integer := 480;
50
51     -- WALL1 - LEFT
52     constant WALL1_X_L: integer := 0;
53     constant WALL1_X_R: integer := 20;
54
55     -- WALL2 - RIGHT
56     constant WALL2_X_L: integer := 619;
57     constant WALL2_X_R: integer := 639;
58
59     -- WALL3 - BOTTOM
60     constant WALL3_X_T: integer := 409;
61     constant WALL3_X_B: integer := 479;
62
63     -- WALL4 - TOP
64     constant WALL4_X_T: integer := 0;
65     constant WALL4_X_B: integer := 20;
66
67     -- Paddle left, right, top, bottom and height -- left & right are constant. Top &
68     -- bottom are signals to allow movement. bar_y_t driven by register below.
69     --constant BAR_X_L: integer := 600;--R2
70     --constant BAR_X_R: integer := 603;--R2

```

```

69  -- signal bar_x_L,bar_x_R: unsigned(9 downto 0); --R2
70  -- signal bar_y_t, bar_y_b: unsigned(9 downto 0);
71  -- constant BAR_Y_SIZE: integer := 72;
72  -- constant BAR_X_SIZE: integer := 8; --R2
73
74  -- -- Reg to track top boundary (x position is fixed)
75  --     signal bar_y_reg, bar_y_next: unsigned( 9 downto 0);
76
77  -- -- Reg to track right boundary --R2
78  --     signal bar_x_reg, bar_x_next: unsigned( 9 downto 0); --R2
79
80  -- Bar moving velocity when a button is pressed -- the amount the bar is moved.
81  -- constant BAR_V: integer:= 4;
82
83  -- Square ball -- ball left, right, top and bottom all vary. Left and top driven by
84  registers below.
85  constant BALL_SIZE: integer := 16; -----CHANGED Increased to 16
86  signal ball_x_l, ball_x_r: unsigned(9 downto 0);
87  signal ball_y_t, ball_y_b: unsigned(9 downto 0);
88
89  -- Reg to track left and top boundary
90  signal ball_x_reg, ball_x_next: unsigned(9 downto 0);
91  signal ball_y_reg, ball_y_next: unsigned(9 downto 0);
92
93  -- reg to track ball speed
94  signal x_delta_reg, x_delta_next: unsigned(9 downto 0);
95  signal y_delta_reg, y_delta_next: unsigned(9 downto 0);
96
97  -- ball movement can be pos or neg
98  constant BALL_V_P: unsigned(9 downto 0):= to_unsigned(asteroid_velocity,10);
99  constant BALL_V_N: unsigned(9 downto 0):= unsigned(to_signed(-
asteroid_velocity,10));
100
101 -- round ball image
102 type rom_type is array(0 to 15) of std_logic_vector(0 to 15); -----Changed
from array(0 to 7)
103 constant BALL_ROM: rom_type:= (
104    "0001111111110000",
105    "0010110011011000",
106    "0111101101110100",
107    "1111011110111110",
108    "1011011101111111",
109    "1111101111011011",
110    "1110110110111111",
111    "1101111001111111",
112    "1101111111011111",
113    "1111011111111111",
114    "1110110011111111",
115    "0111111111111110",
116    "0010111101101100",
117    "0001101111111100",
118    "0000111111111100");
119
120  -- Testing
121  -- 0 is to paint the background and 1 will be to use the ball
122  -- Can make more than one bit to make characters and other sprites.
123
124
125  -- ball is 8x8, the address only needs to be 3 bits then.
126  -- data will first be read as a row
127  -- rom_bit will go to the bit in the row
128  signal rom_addr, rom_col: unsigned(3 downto 0); -----Changed
to 4 bits from unsigned (2 downto 0)
129  signal rom_data: std_logic_vector(15 downto 0); -----Changed
from (7 downto 0)
130  signal rom_bit: std_logic;
131
132  -- object output signals -- new signal to indicate if scan coord is within ball
133  -- signal wall1_on, wall2_on, wall3_on, wall4_on, bar_on, sq_ball_on, rd_ball_on:
std_logic;
134  -- signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, bar_rgb, ball_rgb:
std_logic_vector(2 downto 0);
135  signal wall1_on, wall2_on, wall3_on, wall4_on, sq_ball_on, rd_ball_on: std_logic;
136  signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, ball_rgb: std_logic_vector(2
downto 0);
137
138 ==
=====
```

```

=====
139 begin
140
141 asteroid_on_out <= rd_ball_on;
142 asteroid_rgb_out <= ball_rgb;
143
144 process (clk, reset)
145 begin
146 if (reset = '1') then
147 -- bar_y_reg <= (others => '0');
148 -- bar_x_reg <= (others => '0');--R2
149 ball_x_reg <= (others => '0');
150 ball_y_reg <= (others => '0');
151 x_delta_reg <= ("0000000100");
152 y_delta_reg <= ("0000000100");
153 elsif (clk'event and clk = '1') then
154 -- bar_y_reg <= bar_y_next;
155 -- bar_x_reg <= bar_x_next;--R2
156 ball_x_reg <= ball_x_next;
157 ball_y_reg <= ball_y_next;
158 x_delta_reg <= x_delta_next;
159 y_delta_reg <= y_delta_next;
160 end if;
161 end process;
162
163 --
=====
164 pix_x <= unsigned(pixel_x);
165 pix_y <= unsigned(pixel_y);
166
167 -- Refr_tick: 1-clock tick asserted at start of v_sync, e.g., when the screen is
168 -- refreshed -- speed is 60 Hz
169 refr_tick <= '1' when (pix_y = 1) and (pix_x = 1) else '0';
170
171 --
=====
172 -- -- wall1 left vertical stripe
173 -- wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
174 -- convert pix_x to pix_y to make horizontal
175 -- wall1_rgb <= "011"; -- paddle colors blue
176 -- -- wall2 right vertical stripe
177 -- wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
178 -- wall2_rgb <= "011"; -- paddle colors blue
179 -- -- wall3 left vertical stripe
180 -- wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
181 -- convert pix_x to pix_y to make horizontal
182 -- wall3_rgb <= "011"; -- paddle colors blue
183 -- -- wall4 right vertical stripe
184 -- wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
185 -- wall4_rgb <= "011"; -- paddle colors blue
186
187 --
=====
188 -- -- pixel within paddle
189 -- bar_x_L <= bar_x_reg;--R2
190 -- bar_x_R <= bar_x_L + BAR_X_SIZE - 1;--R2
191 -- bar_y_t <= bar_y_reg;
192 -- bar_y_b <= bar_y_t + BAR_Y_SIZE - 1;
193 -- bar_on <= '1' when (BAR_X_L <= pix_x) and (pix_x <= BAR_X_R) and (bar_y_t <=
194 -- pix_y) and (pix_y <= bar_y_b) else '0';
195 -- bar_rgb <= "100"; -- Red color
196
197 --
=====
198 -- set coordinates of square ball.
199 ball_x_l <= ball_x_reg;
200 ball_y_t <= ball_y_reg;
201 ball_x_r <= ball_x_l + BALL_SIZE - 1;
202 ball_y_b <= ball_y_t + BALL_SIZE - 1;
203
204 -- pixel within square ball
205 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';

```

```

204 -- Map scan coord to ROM addr/col -- use low order three bits of pixel and ball
205   rom_addr <= pix_y(3 downto 0) - ball_y_t(3 downto 0); ----- CHANGED
206   TO 4 BITS
207 -- ROM column
208   rom_col <= pix_x(3 downto 0) - ball_x_l(3 downto 0);----- CHANGED
209   TO 4 BITS
210 -- Get row data
211   rom_data <= BALL_ROM(to_integer(rom_addr));
212
213 -- Get column bit
214   rom_bit <= rom_data(to_integer(rom_col));
215
216 -- Turn ball on only if within square and the ROM bit is 1.
217   rd_ball_on <= '1' when (sq_ball_on = '1') and (rom_bit = '1') else '0';
218   ball_rgb <= "111"; -- WHITE BALL COLOR
219
220 -- Update the ball position 60 times per second.
221   ball_x_next <= ball_x_reg + x_delta_reg when refr_tick = '1' else ball_x_reg;
222   ball_y_next <= ball_y_reg + y_delta_reg when refr_tick = '1' else ball_y_reg;
223
224 -- Set the value of the next ball position according to the boundaries.
225   -- process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_t,
226   ball_y_b, bar_y_t, bar_y_b)
227   process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_t,
228   ball_y_b)
229   begin
230     x_delta_next <= x_delta_reg;
231     y_delta_next <= y_delta_reg;
232
233 -- Ball reached top, make offset positive
234   if ( ball_y_t < WALL4_X_B ) then --MAKE WALL4
235     y_delta_next <= BALL_V_P;
236
237 -- Reached bottom, make negative
238   elsif (ball_y_b > (WALL3_X_T - 1)) then --MAKE WALL3
239     y_delta_next <= BALL_V_N;
240
241 -- Reach wall1, bounce back
242   elsif (ball_x_l <= WALL1_X_R ) then
243     x_delta_next <= BALL_V_F;
244 -- Reach wall2, bounce back
245   elsif (ball_x_r > WALL2_X_L) then
246     x_delta_next <= BALL_V_N;
247
248 -- -- Right corner of ball inside bar
249 --     elsif ((BAR_X_L <= ball_x_r) and (ball_x_r <= BAR_X_R)) then
250 --       if ((bar_y_t <= ball_y_b) and (ball_y_t <= bar_y_b)) then
251 --         x_delta_next <= BALL_V_N;
252 --       end if;
253   end if;
254 end process;
255
256 --
257 =====
258 =====
259 end rtl;
260
261

```

```

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 walls is
21     port(
22         clk, reset: in std_logic;
23         -- btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
24         video_on: in std_logic;
25         pixel_x, pixel_y: in std_logic_vector(9 downto 0);
26         wall1_on_out, wall2_on_out, wall3_on_out, wall4_on_out: out std_logic;
27         wall1_rgb_out, wall2_rgb_out, wall3_rgb_out, wall4_rgb_out: out
28             std_logic_vector(2 downto 0);
29         Left_Wall_Out, Right_Wall_Out, Bottom_Wall_Out, Top_Wall_Out: out
30             std_logic_vector(9 downto 0)
31     );
32 end walls;
33 --
===== =====
34
35 architecture rtl of walls is
36
37     -- Signal used to control speed of ball and how often pushbuttons are checked for
38     -- paddle movement.
39     signal refr_tick: std_logic;
40
41     -- x, y coordinates (0,0 to (639, 479)
42     signal pix_x, pix_y: unsigned(9 downto 0);
43     signal wall1_on, wall2_on, wall3_on, wall4_on: std_logic;
44     signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb: std_logic_vector(2 downto 0);
45     signal Left_Wall, Right_Wall, Bottom_Wall, Top_Wall: std_logic_vector(9 downto 0);
46
47     -- Screen dimensions
48     constant MAX_X: integer := 640;
49     constant MAX_Y: integer := 480;
50
51     -- WALL1 - LEFT
52     constant WALL1_X_L: integer := 0;
53     constant WALL1_X_R: integer := 20;
54
55     -- WALL2 - RIGHT
56     constant WALL2_X_L: integer := 619;
57     constant WALL2_X_R: integer := 639;
58
59     -- WALL3 - BOTTOM
60     constant WALL3_X_T: integer := 409;
61     constant WALL3_X_B: integer := 479;
62
63     -- WALL4 - TOP
64     constant WALL4_X_T: integer := 0;
65     constant WALL4_X_B: integer := 20;
66
67     begin
68         wall1_on_out <= wall1_on;

```

```

68    wall2_on_out <= wall2_on;
69    wall3_on_out <= wall3_on;
70    wall4_on_out <= wall4_on;
71    -- bar_on_out <= rom_bar_on;
72    -- asteroid2_on_out <= rd_ball_on;
73
74    wall1_rgb_out <= wall1_rgb;
75    wall2_rgb_out <= wall2_rgb;
76    wall3_rgb_out <= wall3_rgb;
77    wall4_rgb_out <= wall4_rgb;
78    Left_Wall_Out <= Left_Wall;
79    Right_Wall_Out <= Right_Wall;
80    Bottom_Wall_Out <= Bottom_Wall;
81    Top_Wall_Out <= Top_Wall;
82    Left_Wall <= std_logic_vector(to_unsigned(WALL1_X_R,10));
83    Right_Wall <= std_logic_vector(to_unsigned(WALL2_X_L,10));
84    Bottom_Wall <= std_logic_vector(to_unsigned(WALL3_X_T,10));
85    Top_Wall <= std_logic_vector(to_unsigned(WALL4_X_B,10));
86
87 --
88 =====
89 =====
90     pix_x <= unsigned(pixel_x);
91     pix_y <= unsigned(pixel_y);
92
93 -- Refr_tick: 1-clock tick asserted at start of v_sync, e.g., when the screen is
94 -- refreshed -- speed is 60 Hz
95     refr_tick <= '1' when (pix_y = 1) and (pix_x = 1) else '0';
96
97 --
98 =====
99 =====
100    -- wall1 left vertical stripe
101    wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
102    convert pix_x to pix_y to make horizontal
103    wall1_rgb <= "011"; -- paddle colors blue
104    -- wall2 right vertical stripe
105    wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
106    wall2_rgb <= "011"; -- paddle colors blue
107    -- wall3 left vertical stripe
108    wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
109    convert pix_x to pix_y to make horizontal
110    wall3_rgb <= "011"; -- paddle colors blue
111    -- wall4 right vertical stripe
112    wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
113    wall4_rgb <= "011"; -- paddle colors blue
114
115 end rtl;

```

```

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 walls is
21     port(
22         clk, reset: in std_logic;
23         -- btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
24         video_on: in std_logic;
25         pixel_x, pixel_y: in std_logic_vector(9 downto 0);
26         wall1_on_out, wall2_on_out, wall3_on_out, wall4_on_out: out std_logic;
27         wall1_rgb_out, wall2_rgb_out, wall3_rgb_out, wall4_rgb_out: out
28             std_logic_vector(2 downto 0);
29         Left_Wall_Out, Right_Wall_Out, Bottom_Wall_Out, Top_Wall_Out: out
30             std_logic_vector(9 downto 0)
31     );
32 end walls;
33 --
===== =====
34
35 architecture rtl of walls is
36
37     -- Signal used to control speed of ball and how often pushbuttons are checked for
38     -- paddle movement.
39     signal refr_tick: std_logic;
40
41     -- x, y coordinates (0,0 to (639, 479)
42     signal pix_x, pix_y: unsigned(9 downto 0);
43     signal wall1_on, wall2_on, wall3_on, wall4_on: std_logic;
44     signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb: std_logic_vector(2 downto 0);
45     signal Left_Wall, Right_Wall, Bottom_Wall, Top_Wall: std_logic_vector(9 downto 0);
46
47     -- Screen dimensions
48     constant MAX_X: integer := 640;
49     constant MAX_Y: integer := 480;
50
51     -- WALL1 - LEFT
52     constant WALL1_X_L: integer := 0;
53     constant WALL1_X_R: integer := 20;
54
55     -- WALL2 - RIGHT
56     constant WALL2_X_L: integer := 619;
57     constant WALL2_X_R: integer := 639;
58
59     -- WALL3 - BOTTOM
60     constant WALL3_X_T: integer := 409;
61     constant WALL3_X_B: integer := 479;
62
63     -- WALL4 - TOP
64     constant WALL4_X_T: integer := 0;
65     constant WALL4_X_B: integer := 20;
66
67     begin
68         wall1_on_out <= wall1_on;

```

```

68  wall2_on_out <= wall2_on;
69  wall3_on_out <= wall3_on;
70  wall4_on_out <= wall4_on;
71  -- bar_on_out <= rom_bar_on;
72  -- asteroid2_on_out <= rd_ball_on;
73
74  wall1_rgb_out <= wall1_rgb;
75  wall2_rgb_out <= wall2_rgb;
76  wall3_rgb_out <= wall3_rgb;
77  wall4_rgb_out <= wall4_rgb;
78  Left_Wall_Out <= Left_Wall;
79  Right_Wall_Out <= Right_Wall;
80  Bottom_Wall_Out <= Bottom_Wall;
81  Top_Wall_Out <= Top_Wall;
82  Left_Wall <= std_logic_vector(to_unsigned(WALL1_X_R,10));
83  Right_Wall <= std_logic_vector(to_unsigned(WALL2_X_L,10));
84  Bottom_Wall <= std_logic_vector(to_unsigned(WALL3_X_T,10));
85  Top_Wall <= std_logic_vector(to_unsigned(WALL4_X_B,10));
86
87 --
88 =====
89 =====
90  pix_x <= unsigned(pixel_x);
91  pix_y <= unsigned(pixel_y);
92
93  -- Refr_tick: 1-clock tick asserted at start of v_sync, e.g., when the screen is
94  -- refreshed -- speed is 60 Hz
95  refr_tick <= '1' when (pix_y = 1) and (pix_x = 1) else '0';
96
97  --
98  =====
99  =====
100 -- wall1 left vertical stripe
101  wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
102  convert pix_x to pix_y to make horizontal
103  wall1_rgb <= "011"; -- paddle colors blue
104 -- wall2 right vertical stripe
105  wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
106  wall2_rgb <= "011"; -- paddle colors blue
107 -- wall3 left vertical stripe
108  wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
109  convert pix_x to pix_y to make horizontal
110  wall3_rgb <= "011"; -- paddle colors blue
111 -- wall4 right vertical stripe
112  wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
113  wall4_rgb <= "011"; -- paddle colors blue
114
115 end rtl;

```

```

1 -----
2 -- Company: University of New Mexico
3 -- Engineer: Professor Jim Plusquellic, Copyright Univ. of New Mexico
4 --
5 -- Create Date:
6 -- Design Name:
7 -- Module Name: LoadUnLoadMem - Behavioral
8 -- Project Name:
9 -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21 -- LoadUnLoadMem securely transfers data into or out of PNL_BRAM using GPIO registers
22
23 library IEEE;
24 use IEEE.STD_LOGIC_1164.ALL;
25 use IEEE.NUMERIC_STD.all;
26
27 library work;
28 use work.DataTypes_pkg.all;
29
30 entity LoadUnLoadMem is
31     port(
32         Clk: in std_logic;
33         RESET: in std_logic;
34         start: in std_logic;
35         ready: out std_logic;
36         load_unload: in std_logic;
37         stopped: out std_logic;
38         continue: in std_logic;
39         done: in std_logic;
40         base_address: in std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
41         upper_limit: in std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
42         CP_in_word: in std_logic_vector(WORD_SIZE_NB-1 downto 0);
43         CP_out_word: out std_logic_vector(WORD_SIZE_NB-1 downto 0);
44         PNL_BRAM_addr: out std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
45         PNL_BRAM_din: out std_logic_vector(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
46         PNL_BRAM_dout: in std_logic_vector(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
47         PNL_BRAM_we: out std_logic_vector(0 to 0)
48     );
49 end LoadUnLoadMem;
50
51 architecture beh of LoadUnLoadMem is
52     type state_type is (idle, load_mem, unload_mem, wait_load_unload, wait_done);
53     signal state_reg, state_next: state_type;
54
55     signal ready_reg, ready_next: std_logic;
56
57     signal PNL_BRAM_addr_reg, PNL_BRAM_addr_next: unsigned(PNL_BRAM_ADDR_SIZE_NB-1
58     downto 0);
59     signal PNL_BRAM_upper_limit_reg, PNL_BRAM_upper_limit_next:
60     unsigned(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
61
62     begin
63
64     =====
65     =====
66     -- State and register logic
67     =====
68     =====
69
70     process(Clk, RESET)
71     begin
72         if ( RESET = '1' ) then
73             state_reg <= idle;
74             ready_reg <= '1';
75             PNL_BRAM_addr_reg <= (others=>'0');
76             PNL_BRAM_upper_limit_reg <= (others=>'0');
77         elsif ( Clk'event and Clk = '1' ) then

```

```

73     state_reg <= state_next;
74     ready_reg <= ready_next;
75     PNL_BRAM_addr_reg <= PNL_BRAM_addr_next;
76     PNL_BRAM_upper_limit_reg <= PNL_BRAM_upper_limit_next;
77   end if;
78 end process;
79
80 --
===== =====
81 -- Combo logic
82 --
===== =====
83 process (state_reg, start, ready_reg, load_unload, PNL_BRAM_addr_reg,
84 PNL_BRAM_upper_limit_reg,
85   PNL_BRAM_dout, CP_in_word, continue, base_address, upper_limit, done)
86 begin
87   state_next <= state_reg;
88   ready_next <= ready_reg;
89
90   PNL_BRAM_addr_next <= PNL_BRAM_addr_reg;
91   PNL_BRAM_upper_limit_next <= PNL_BRAM_upper_limit_reg;
92
93   PNL_BRAM_we <= "0";
94   PNL_BRAM_din <= (others=>'0');
95   CP_out_word <= (others=>'0');
96
97   stopped <= '0';
98
99   case state_reg is
100  --
101    when idle =>
102      ready_next <= '1';
103
104    if ( start = '1' ) then
105      ready_next <= '0';
106
107  -- Latch the 'base_address' and 'upper_limit' at the instant 'start' is asserted.
108  -- NOTE: These signals MAY BE SET
109  -- BACK TO all 0's after the 'start' signal is received.
110  -- PNL_BRAM_addr_next <= unsigned(base_address);
111  -- PNL_BRAM_upper_limit_next <= unsigned(upper_limit);
112
113  if ( load_unload = '0' ) then
114    state_next <= load_mem;
115  else
116    state_next <= unload_mem;
117  end if;
118 end if;
119
120  --
121  -- Write value to memory location
122  -- when load_mem =>
123
124  -- Signal C program that we are ready to receive a word. Once ready ('continue'
125  -- becomes '1'), transfer and complete handshake.
126  stopped <= '1';
127  if ( done = '0' ) then
128    if ( continue = '1' ) then
129      PNL_BRAM_we <= "1";
130      PNL_BRAM_din <= (PNL_BRAM_DBITS_WIDTH_NB-1 downto WORD_SIZE_NB =>
131        '0') & CP_in_word;
132
133  -- Wait handshake signals
134  -- state_next <= wait_load_unload;
135  -- end if;
136
137  -- Handle case where C program has nothing to store.
138  else
139    state_next <= wait_done;
140
141  -- Get value at memory location
142  -- when unload_mem =>

```

```

143 -- Put the PNL BRAM word on CP_out_word. Do NOT do this by default for security
144   reasons.
145   CP_out_word <= PNL_BRAM_dout(WORD_SIZE_NB-1 downto 0);
146
146 -- Signal C program that we are ready to deliver a word. Once it reads the word, it
147   sets 'continue' to '1'.
147   stopped <= '1';
148   if ( continue = '1' ) then
149
150 -- Wait handshake signals
151   state_next <= wait_load_unload;
152 end if;
153
154 -- Handle case where C program does not want to read any data.
155   if ( done = '1' ) then
156     state_next <= wait_done;
157   end if;
158
159 -- =====
160 -- Complete handshake and update addresses
161 when wait_load_unload =>
162
163 -- C program holds 'continue' at 1 until it sees 'stopped' go to 0, and then it
164   writes a '0' to continue. It also writes
164 -- 'done' with a '1' when last transfer is made.
165   if ( continue = '0' ) then
166
167 -- Done collecting C program transmitted words. Force a finish if the upper limit has
168   been reached. This will protect the memory
168 -- from overruns (reading or writing).
169   if ( done = '1' ) then
170     state_next <= wait_done;
171   elsif ( PNL_BRAM_addr_reg = PNL_BRAM_upper_limit_reg ) then
172     state_next <= idle;
173   else
174     PNL_BRAM_addr_next <= PNL_BRAM_addr_reg + 1;
175     if ( load_unload = '0' ) then
176       state_next <= load_mem;
177     else
178       state_next <= unload_mem;
179     end if;
180   end if;
181 end if;
182
183 -- =====
184 -- Wait for 'done' to return to 0 before returning to idle, if it was set by the C
184   program to exit early.
185 when wait_done =>
186   if ( done = '0' ) then
187     state_next <= idle;
188   end if;
189
190 end case;
191 end process;
192
193 -- Use 'look-ahead' signal for BRAM address.
194 PNL_BRAM_addr <= std_logic_vector(PNL_BRAM_addr_next);
195 ready <= ready_reg;
196
197 end beh;
198
199

```

```

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
37         unsigned(9 downto 0)
38     );
39 end missile;
40
41 --
===== =====
42 --
===== =====
43
44 architecture rtl of missile is
45
46 -- Signal used to control speed of ball and how often pushbuttons are checked for
47 -- paddle movement.
48     signal refr_tick: std_logic;
49
50 -- x, y coordinates (0,0 to (639, 479)
51     signal pix_x, pix_y: unsigned(9 downto 0);
52
53 -- WALL1 - LEFT
54     constant WALL1_X_L: integer := 0;
55     constant WALL1_X_R: integer := 20;
56
57 -- WALL2 - RIGHT
58     constant WALL2_X_L: integer := 619;
59     constant WALL2_X_R: integer := 639;
60
61 -- WALL3 - BOTTOM
62     constant WALL3_X_T: integer := 409;
63     constant WALL3_X_B: integer := 479;
64
65 -- WALL4 - TOP
66     constant WALL4_X_T: integer := 0;
67     constant WALL4_X_B: integer := 20;

```

```

68 -- Square ball -- ball left, right, top and bottom all vary. Left and top driven by
69 registers below.
70 constant BALL_SIZE: integer := 16; -----CHANGED Increased to 16
71 signal ball_x_l, ball_x_r: unsigned(9 downto 0);
72 signal ball_y_t, ball_y_b: unsigned(9 downto 0);
73
74 -- Reg to track left and top boundary
75 signal ball_x_reg, ball_x_next, ball_x_next_mux: unsigned(9 downto 0);
76 signal ball_y_reg, ball_y_next, ball_y_next_mux: unsigned(9 downto 0);
77
78 -- reg to track ball speed
79 signal x_delta_reg, x_delta_next: unsigned(9 downto 0);
80 signal y_delta_reg, y_delta_next: unsigned(9 downto 0);
81
82 -- ball movement can be pos or neg
83 constant BALL_V_P: unsigned(9 downto 0):= to_unsigned(missile_velocity,10);
84 constant BALL_V_N: unsigned(9 downto 0):= unsigned(to_signed(-
missile_velocity,10));
85 constant BALL_V_NULL: unsigned(9 downto 0):= to_unsigned(0,10);
86
87 -- round ball image
88 type rom_type is array(0 to 15) of std_logic_vector(0 to 15); -----Changed
from array(0 to 7)
89 constant BALL_ROM_UP: rom_type:= (
90 "000000011000000",
91 "110000011000001",
92 "110000000000001",
93 "000000000000000",
94 "000000000000000",
95 "000000011000000",
96 "110000011000001",
97 "110000011000001",
98 "110000011000001",
99 "110000011000001",
100 "110000000000001",
101 "000000000000000",
102 "000000000000000",
103 "000000000000000",
104 "000000000000000");
105 constant BALL_ROM_DOWN: rom_type:= (
106 "000000000000000",
107 "000000000000000",
108 "000000000000000",
109 "000000000000000",
110 "110000011000001",
111 "110000000000001",
112 "110000000000001",
113 "110000000000001",
114 "110000000000001",
115 "110000000000001",
116 "000000011000000",
117 "000000000000000",
118 "000000000000000",
119 "110000000000001",
120 "110000011000001",
121 "000000011000000");
122 constant BALL_ROM_LEFT: rom_type:= (
123 "011000111111000",
124 "011000111111000",
125 "000000000000000",
126 "000000000000000",
127 "000000000000000",
128 "000000000000000",
129 "000000000000000",
130 "110001111100000",
131 "110001111100000",
132 "000000000000000",
133 "000000000000000",
134 "000000000000000",
135 "000000000000000",
136 "000000000000000",
137 "011000111111000",
138 "011000111111000");
139 constant BALL_ROM_RIGHT: rom_type:= (
140 "000011111100110",
141 "000011111100110",
142 "000000000000000",
143 "000000000000000",

```

```

144      "0000000000000000",
145      "0000000000000000",
146      "0000000000000000",
147      "0000011111100011",
148      "0000011111100011",
149      "0000000000000000",
150      "0000000000000000",
151      "0000000000000000",
152      "0000000000000000",
153      "0000000000000000",
154      "0000111111000110",
155      "0000111111000110");
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
165      to 4 bits from unsigned (2 downto 0)
166      signal rom_data: std_logic_vector(15 downto 0); -----Changed
167      from (7 downto 0)
168      signal rom_bit: std_logic;
169
170      -- object output signals -- new signal to indicate if scan coord is within ball
171      signal sq_ball_on, rd_ball_on: std_logic;
172      signal ball_rgb: std_logic_vector(2 downto 0);
173      signal fire_ready_reg, fire_ready_next: std_logic;
174      signal rom_selector_reg, rom_selector_next: std_logic_vector(2 downto 0);
175
176      --
177      =====
178      begin
179
180          missile_on_out <= rd_ball_on;
181          missile_rgb_out <= ball_rgb;
182          fire_ready_out <= fire_ready_next;
183
184          with fire_out select
185              ball_x_next_mux <= bar_x_reg_out when '0',
186              ball_x_next when '1';
187
188          with fire_out select
189              ball_y_next_mux <= bar_y_reg_out when '0',
190              ball_y_next when '1';
191
192          process (clk, reset)
193              begin
194                  if (reset = '1') then
195                      ball_x_reg <= bar_x_reg_out;
196                      ball_y_reg <= bar_y_reg_out;
197                      x_delta_reg <= ("0000000000");
198                      y_delta_reg <= ("0000000000");
199                      fire_ready_reg <= '0';
200                      rom_selector_reg <= "111";
201
202                  elsif (clk'event and clk = '1') then
203                      ball_x_reg <= ball_x_next_mux;
204                      ball_y_reg <= ball_y_next_mux;
205                      x_delta_reg <= x_delta_next;
206                      y_delta_reg <= y_delta_next;
207                      fire_ready_reg <= fire_ready_next;
208                      rom_selector_reg <= rom_selector_next;
209
210                  end if;
211          end process;
212
213
214      pix_x <= unsigned(pixel_x);
215      pix_y <= unsigned(pixel_y);
216
217      -- Refr_tick: 1-clock tick asserted at start of v_sync, e.g., when the screen is
218      -- refreshed -- speed is 60 Hz
219      -- 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 =
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
224        fire_button_ready_signal!!;
225        rom_selector_next <= rom_selector_out;
226    end if;
227 end process;
228 --
229 =====
230 -- set coordinates of square ball.
231 ball_x_l <= ball_x_reg;
232 ball_y_t <= ball_y_reg;
233 ball_x_r <= ball_x_l + BALL_SIZE - 1;
234 ball_y_b <= ball_y_t + BALL_SIZE - 1;
235 --
236 -- pixel within square ball
237 sq_ball_on <= '1' when (ball_x_l <= pix_x) and (pix_x <= ball_x_r) and (ball_y_t
238 <= pix_y) and (pix_y <= ball_y_b) else '0';
239 --
240 -- Map scan coord to ROM addr/col -- use low order three bits of pixel and ball
241 positions. ROM row
242 rom_addr <= pix_y(3 downto 0) - ball_y_t(3 downto 0); ----- CHANGED
243 TO 4 BITS
244 --
245 -- ROM column
246 rom_col <= pix_x(3 downto 0) - ball_x_l(3 downto 0);----- CHANGED
247 TO 4 BITS
248 --
249 -- Get row data
250 process(fire_out,rom_selector_out)
251 begin
252    if (fire_out = '0') then
253        if (rom_selector_out = "110") then
254            rom_data <= BALL_ROM_UP(to_integer(rom_addr));
255        elsif (rom_selector_out = "000") then
256            rom_data <= BALL_ROM_DOWN(to_integer(rom_addr));
257        elsif (rom_selector_out = "100") then
258            rom_data <= BALL_ROM_LEFT(to_integer(rom_addr));
259        elsif (rom_selector_out = "010") then
260            rom_data <= BALL_ROM_RIGHT(to_integer(rom_addr));
261    end if;
262    elsif (fire_out = '1') then
263        if (rom_selector_reg = "110") then
264            rom_data <= BALL_ROM_UP(to_integer(rom_addr));
265        elsif (rom_selector_reg = "000") then
266            rom_data <= BALL_ROM_DOWN(to_integer(rom_addr));
267        elsif (rom_selector_reg = "100") then
268            rom_data <= BALL_ROM_LEFT(to_integer(rom_addr));
269        elsif (rom_selector_reg = "010") then
270            rom_data <= BALL_ROM_RIGHT(to_integer(rom_addr));
271    end if;
272 end if;
273 end process;
274 --
275 -- Get column bit
276 rom_bit <= rom_data(to_integer(rom_col));
277 --
278 -- Turn ball on only if within square and the ROM bit is 1.
279 rd_ball_on <= '1' when (sq_ball_on = '1') and (rom_bit = '1') else '0';
280 ball_rgb <= "111"; -- WHITE BALL COLOR
281 --
282 -- Update the ball position 60 times per second.
283 ball_x_next <= ball_x_reg + x_delta_reg when refr_tick = '1' else ball_x_reg;
284 ball_y_next <= ball_y_reg + y_delta_reg when refr_tick = '1' else ball_y_reg;
285 --
286 -- 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,
285    fire_ready_reg, rom_selector_reg)
286    begin
287    --    x_delta_next <= x_delta_reg; -- don't move
288    --    y_delta_next <= y_delta_reg; -- don't move
289    --    fire_ready_next <= fire_ready_reg;
290
291    --    if (rom_selector_reg = "110" and ball_y_t >= WALL4_X_B and ball_y_t < 479 )
292    then -- DOWN
293    --        y_delta_next <= BALL_V_N;
294    --        x_delta_next <= BALL_V_NULL;
295    --    elsif (rom_selector_reg = "110" and ball_y_t = WALL4_X_B and ball_y_t < 479
296 ) then -- DOWN
297    --        y_delta_next <= BALL_V_NULL;
298    --        x_delta_next <= BALL_V_NULL;
299    --        fire_ready_next <= '1';
300
301    --    elsif (rom_selector_reg = "000" and ball_y_b <= WALL3_X_T and ball_y_b > 0)
302    then -- UP
303    --        y_delta_next <= BALL_V_P;
304    --        x_delta_next <= BALL_V_NULL;
305    --    elsif (rom_selector_reg = "000" and ball_y_b = WALL3_X_T and ball_y_b > 0)
306    then -- UP
307    --        y_delta_next <= BALL_V_NULL;
308    --        x_delta_next <= BALL_V_NULL;
309    --        fire_ready_next <= '1';
310
311    --    elsif (rom_selector_reg = "010" and ball_x_l >= WALL1_X_R and ball_x_l > 0)
312    then -- RIGHT
313    --        x_delta_next <= BALL_V_N;
314    --        y_delta_next <= BALL_V_NULL;
315    --    elsif (rom_selector_reg = "010" and ball_x_l = WALL1_X_R and ball_x_l > 0)
316    then -- RIGHT
317    --        y_delta_next <= BALL_V_NULL;
318    --        x_delta_next <= BALL_V_NULL;
319    --        fire_ready_next <= '1';
320
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,
325    fire_ready_reg, rom_selector_reg)
326    begin
327    x_delta_next <= x_delta_reg; -- don't move
328    y_delta_next <= y_delta_reg; -- don't move
329    fire_ready_next <= fire_ready_reg;
330
331    if (rom_selector_reg = "110" and ball_y_t >= WALL4_X_B and ball_y_t < 479 )
332    then -- DOWN
333    y_delta_next <= BALL_V_N;
334    x_delta_next <= BALL_V_NULL;
335
336    if ( ball_y_t <= WALL4_X_B ) then
337        y_delta_next <= BALL_V_NULL;
338        x_delta_next <= BALL_V_NULL;
339        fire_ready_next <= '1';
340    else
341        fire_ready_next <= '0';
342    end if;
343
344    elsif (rom_selector_reg = "000" and ball_y_b <= WALL3_X_T and ball_y_b > 0)
345    then -- UP
346    y_delta_next <= BALL_V_P;
347    x_delta_next <= BALL_V_NULL;
348
349    if (ball_y_b >= WALL3_X_T) then
350        y_delta_next <= BALL_V_NULL;

```

```

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 -----
```

```

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--      EXCEPT FOR PARTS OF LINES 236 and 241 SPECIFICALLY THE COMPARISON
12--      ORIENTATION
13--      THESE PARTS WERE CREATED BY THE FOLLOWING
14--      Engineer: Anindya Bal
15-----
16
17--
18=====
19=====
20=====
21=====
22
23library ieee;
24use ieee.std_logic_1164.all;
25use ieee.numeric_std.all;
26
27entity space_ship is
28  port(
29    clk, reset: in std_logic;
30    btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
31    sw: in std_logic_vector(0 downto 0);
32    video_on: in std_logic;
33    pixel_x, pixel_y: in std_logic_vector(9 downto 0);
34    bar_on_out: out std_logic;
35    bar_rgb_out: out std_logic_vector(2 downto 0);
36    bar_x_reg_out, bar_y_reg_out: out unsigned( 9 downto 0 );
37    rom_selector_out: out std_logic_vector(2 downto 0);
38    graph_rgb: out std_logic_vector(2 downto 0);
39    fire_out,fire_btn_signal_out: out std_logic;
40    fire_ready_out: in std_logic
41  );
42end space_ship;
43
44=====
45
46architecture rtl of space_ship is
47
48  -- Signal used to control speed of ball and how often pushbuttons are checked for
49  -- paddle movement.
50  signal refr_tick: std_logic;
51
52  -- x, y coordinates (0,0 to (639, 479)
53  signal pix_x, pix_y: unsigned(9 downto 0);
54
55  -- Screen dimensions
56  constant MAX_X: integer := 640;
57  constant MAX_Y: integer := 480;
58
59  -- WALL1 - LEFT
60  constant WALL1_X_L: integer := 0;
61  constant WALL1_X_R: integer := 20;
62
63  -- WALL2 - RIGHT
64  constant WALL2_X_L: integer := 619;
65  constant WALL2_X_R: integer := 639;
66
67  -- WALL3 - BOTTOM
68  constant WALL3_X_T: integer := 409;
69  constant WALL3_X_B: integer := 479;
70
71  -- WALL4 - TOP

```



```

144     "001111111100000");
145
146     constant SPACESHIP_LEFT_ROM: spaceship_rom_type:= (
147         "000001111111100",
148         "000001111111100",
149         "0000000001100000",
150         "0000000001100011",
151         "0000000001100010",
152         "0000000001111011",
153         "000111111111101",
154         "1111111001100010",
155         "1111111001100010",
156         "000111111111101",
157         "0000000001111011",
158         "0000000001100010",
159         "0000000001100011",
160         "0000000001100000",
161         "000001111111100",
162         "000001111111100");
163
164     signal spaceship_rom_addr, spaceship_rom_col: unsigned(3 downto 0); -----
165     -----Changed to 4 bits from unsigned (2 downto 0)
166     signal spaceship_rom_data: std_logic_vector(15 downto 0); -----
167     -Changed from (7 downto 0)
168     signal spaceship_rom_bit: std_logic;
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214

```

```

215
216     end if;
217 end process;
218
219
220 -- Process bar movement requests
221 process( bar_y_reg, bar_y_b, bar_y_t,bar_x_reg,bar_x_L,bar_x_R, refr_tick, btn,
222 rom_selector)
223 begin
224     bar_y_next <= bar_y_reg; -- no move
225     bar_x_next <= bar_x_reg; -- no move--R2
226
227     if ( refr_tick = '1' ) then
228         --SW0 DOWN and BTN2 1 --FIRE DOWN NOT AT EDGE
229         if ( btn(2) = '1' and sw(0) = '0' and bar_y_b < (WALL3_X_T - 1 - BAR_V)) then -
230             -CHANGE TO WALL PARAMETERS
231             bar_y_next <= bar_y_reg + BAR_V; -- move down
232             rom_selector <= "000";
233
234         --SW0 UP and_BTN2 1 --FIRE UP NOT AT EDGE
235         elsif ( btn(2) = '1' and sw(0) = '1' and bar_y_t > (WALL4_X_B - 1 - BAR_V))
236     then
237             bar_y_next <= bar_y_reg - BAR_V; -- move up
238             rom_selector <= "110";
239
240         -- if btn 0 pressed and bar not at RIGHT yet --R2
241         elsif (btn(1) = '1' and bar_x_R < (WALL2_X_L - 1 - BAR_V)) then
242             bar_x_next <= bar_x_reg + BAR_V; -- move RIGHT--R2
243             rom_selector <= "100";
244
245         -- if btn 0 NOT pressed and bar not at yet
246         elsif (btn(3) = '1' and bar_x_L > (WALL1_X_R - 1 - BAR_V)) then
247             bar_x_next <= bar_x_reg - BAR_V;--move LEFT
248             rom_selector <= "010";
249
250         end if;
251     end if;
252 end process;
253
254 with rom_selector select
255     spaceship_rom_data <= SPACESHIP_UP_ROM(to_integer(spaceship_rom_addr)) when
256     "110",
257             SPACESHIP_DOWN_ROM(to_integer(spaceship_rom_addr)) when
258     "000",
259             SPACESHIP_LEFT_ROM(to_integer(spaceship_rom_addr)) when
260     "100",
261             SPACESHIP_RIGHT_ROM(to_integer(spaceship_rom_addr)) when
262 others;
263
264     spaceship_rom_bit <= spaceship_rom_data(to_integer(spaceship_rom_col));-- Get
265 column bit
266
267 =====
268 =====
269     pix_x <= unsigned(pixel_x);
270     pix_y <= unsigned(pixel_y);
271
272 -- Refr_tick: 1-clock tick asserted at start of v_sync, e.g., when the screen is
273 -- refreshed -- speed is 60 Hz
274     refr_tick <= '1' when (pix_y = 1) and (pix_x = 1) else '0';
275
276 --
277 =====
278 =====
279 -- pixel within paddle
280
281     bar_x_L <= bar_x_reg;--R2
282     bar_y_t <= bar_y_reg;
283     bar_x_R <= bar_x_L + BAR_SIZE - 1;--R2
284     bar_y_b <= bar_y_t + BAR_SIZE - 1;
285     bar_on <= '1' when (BAR_X_L <= pix_x) and (pix_x <= BAR_X_R) and (bar_y_t <=
286 pix_y) and (pix_y <= bar_y_b) else '0';
287     bar_rgb <= "100"; -- Red color
288     -- Map scan coord to ROM addr/col -- use low order three bits of pixel and ball
289     positions. ROM row
290     spaceship_rom_addr <= pix_y(3 downto 0) - bar_y_t(3 downto 0); -----
291     ---- CHANGED TO 4 BITS
292     -- ROM column

```

```
277      spaceship_rom_col <= pix_x(3 downto 0) - bar_x_L(3 downto 0);-----  
-- CHANGED TO 4 BITS  
278      -- Get row data  
279      --    spaceship_rom_data <= SPACESHIP_UP_ROM(to_integer(spaceship_rom_addr));  
280      -- -- Get column bit  
281      --    spaceship_rom_bit <= spaceship_rom_data(to_integer(spaceship_rom_col));  
282      -- Turn ball on only if within square and the ROM bit is 1.  
283      rom_bar_on <= '1' when (bar_on = '1') and (spaceship_rom_bit = '1') else '0';  
284      bar_rgb <= "111"; -- WHITE BALL COLOR  
285  
286  
287  
288  
289 end rtl;
```

```

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 walls is
21     port(
22         clk, reset: in std_logic;
23         -- btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
24         video_on: in std_logic;
25         pixel_x, pixel_y: in std_logic_vector(9 downto 0);
26         wall1_on_out, wall2_on_out, wall3_on_out, wall4_on_out: out std_logic;
27         wall1_rgb_out, wall2_rgb_out, wall3_rgb_out, wall4_rgb_out: out
28             std_logic_vector(2 downto 0);
29         Left_Wall_Out, Right_Wall_Out, Bottom_Wall_Out, Top_Wall_Out: out
30             std_logic_vector(9 downto 0)
31     );
32 end walls;
33 --
===== =====
34
35 architecture rtl of walls is
36
37     -- Signal used to control speed of ball and how often pushbuttons are checked for
38     -- paddle movement.
39     signal refr_tick: std_logic;
40
41     -- x, y coordinates (0,0 to (639, 479)
42     signal pix_x, pix_y: unsigned(9 downto 0);
43     signal wall1_on, wall2_on, wall3_on, wall4_on: std_logic;
44     signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb: std_logic_vector(2 downto 0);
45     signal Left_Wall, Right_Wall, Bottom_Wall, Top_Wall: std_logic_vector(9 downto 0);
46
47     -- Screen dimensions
48     constant MAX_X: integer := 640;
49     constant MAX_Y: integer := 480;
50
51     -- WALL1 - LEFT
52     constant WALL1_X_L: integer := 0;
53     constant WALL1_X_R: integer := 20;
54
55     -- WALL2 - RIGHT
56     constant WALL2_X_L: integer := 619;
57     constant WALL2_X_R: integer := 639;
58
59     -- WALL3 - BOTTOM
60     constant WALL3_X_T: integer := 409;
61     constant WALL3_X_B: integer := 479;
62
63     -- WALL4 - TOP
64     constant WALL4_X_T: integer := 0;
65     constant WALL4_X_B: integer := 20;
66
67     begin
68         wall1_on_out <= wall1_on;

```

```

68  wall2_on_out <= wall2_on;
69  wall3_on_out <= wall3_on;
70  wall4_on_out <= wall4_on;
71  -- bar_on_out <= rom_bar_on;
72  -- asteroid2_on_out <= rd_ball_on;
73
74  wall1_rgb_out <= wall1_rgb;
75  wall2_rgb_out <= wall2_rgb;
76  wall3_rgb_out <= wall3_rgb;
77  wall4_rgb_out <= wall4_rgb;
78  Left_Wall_Out <= Left_Wall;
79  Right_Wall_Out <= Right_Wall;
80  Bottom_Wall_Out <= Bottom_Wall;
81  Top_Wall_Out <= Top_Wall;
82  Left_Wall <= std_logic_vector(to_unsigned(WALL1_X_R,10));
83  Right_Wall <= std_logic_vector(to_unsigned(WALL2_X_L,10));
84  Bottom_Wall <= std_logic_vector(to_unsigned(WALL3_X_T,10));
85  Top_Wall <= std_logic_vector(to_unsigned(WALL4_X_B,10));
86
87 --
88 =====
89 =====
90  pix_x <= unsigned(pixel_x);
91  pix_y <= unsigned(pixel_y);
92
93  -- Refr_tick: 1-clock tick asserted at start of v_sync, e.g., when the screen is
94  -- refreshed -- speed is 60 Hz
95  refr_tick <= '1' when (pix_y = 1) and (pix_x = 1) else '0';
96
97  --
98  =====
99  =====
100 -- wall1 left vertical stripe
101  wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
102  convert pix_x to pix_y to make horizontal
103  wall1_rgb <= "011"; -- paddle colors blue
104 -- wall2 right vertical stripe
105  wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
106  wall2_rgb <= "011"; -- paddle colors blue
107 -- wall3 left vertical stripe
108  wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
109  convert pix_x to pix_y to make horizontal
110  wall3_rgb <= "011"; -- paddle colors blue
111 -- wall4 right vertical stripe
112  wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
113  wall4_rgb <= "011"; -- paddle colors blue
114
115 end rtl;

```