Final Project Report
FPGA Videogame

Intermediate Logic Design

Professor: Jim, Plusquellic
Student: Montoya, Zachary

Submitted: December 15th, 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 Zybo 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

Date: 12-07-22	**Revision Number: 4**
Title: Multiple Asteroid Instantiations via modules, missiles, ship ROM, and the beginnings of text	
Authors: Zachary Montoya	
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

```
##Switches
set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L19N_T3_VREF_35 Sch=sw[0]
#set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { sw_g }]; #IO_L24P_T3_34 Sch=sw[1]
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { sw_b }]; #IO_L4N_T0_34 Sch=sw[2]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=sw[3]


##Buttons
set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }]; #IO_L12N_T1_MRCC_35 Sch=btn[0]
set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L24N_T3_34 Sch=btn[1]
set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; #IO_L10P_T1_AD11P_35 Sch=btn[2]
set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L7P_T1_34 Sch=btn[3]
```

Figure 3 – Additional game boundaries

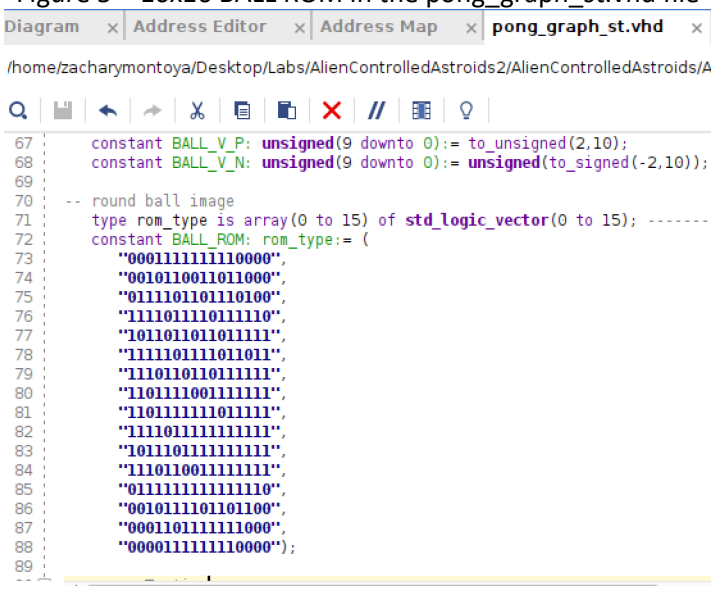
```
-- Process bar movement requests
   process( bar_y_reg, bar_y_b, bar_y_t,bar_x_reg,bar_x_L,bar_x_R, refr_tick, btn)
      begin
      bar_y_next <= bar_y_reg; -- no move
      bar_x_next <= bar_x_reg; -- no move--R2

      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:

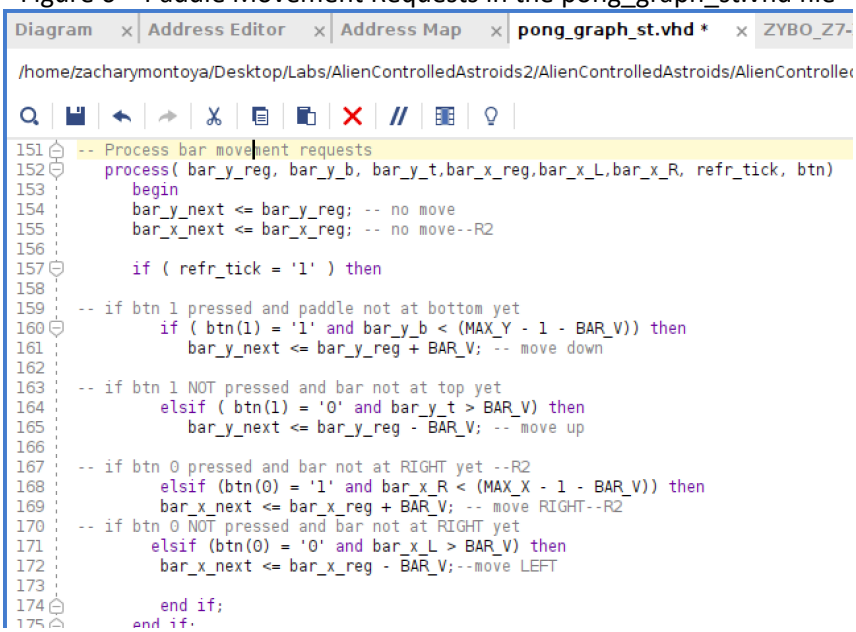
Figure 4 - Lines 54 of pong_graph_st.vhd file

Figure 5 – 16x16 BALL ROM in the pong_graph_st.vhd file



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



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

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

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

--
================================================================================================
--
================================================================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

library work;
use work.DataTypes_pkg.all;

entity Top is
   port (
      Clk: in std_logic;
      reset : in STD_LOGIC;
      GPIO_Ins: in std_logic_vector(31 downto 0);
      GPIO_Outs: out std_logic_vector(31 downto 0);
      PNL_BRAM_addr: out std_logic_vector (12 downto 0);
      PNL_BRAM_din: out std_logic_vector (15 downto 0);
      PNL_BRAM_dout: in std_logic_vector (15 downto 0);
      PNL_BRAM_we: out std_logic_vector (0 to 0);
      hdmi_red : out STD_LOGIC_VECTOR ( 7 downto 0 );
      hdmi_green : out STD_LOGIC_VECTOR ( 7 downto 0 );
      hdmi_blue : out STD_LOGIC_VECTOR ( 7 downto 0 );
      hdmi_hsync : out STD_LOGIC;
      hdmi_vsync : out STD_LOGIC;
      hdmi_enable : out STD_LOGIC;
      btn: in std_logic_vector(3 downto 0); --R2 increasing the array from 2 to 3.
      sw: in std_logic_vector( 0 downto 0) --R2 increasing the array from 2 to 3.
  --    DEBUG_IN: in std_logic;
   --   DEBUG_OUT: out std_logic
      );
end Top;

architecture beh of Top is

-- GPIO INPUT BIT ASSIGNMENTS
   constant IN_CP_RESET: integer := 31;
   constant IN_CP_START: integer := 30;
   constant IN_CP_LM_ULM_LOAD_UNLOAD: integer := 26;
   constant IN_CP_LM_ULM_DONE: integer := 25;
   constant IN_CP_HANDSHAKE: integer := 24;

-- GPIO OUTPUT BIT ASSIGNMENTS
   constant OUT_SM_READY: integer := 31;
   constant OUT_SM_HANDSHAKE: integer := 28;

-- Signal declarations
   signal RESET_final: std_logic;

   signal pixel_x: unsigned(10 downto 0);
   signal pixel_y: unsigned(9 downto 0);
   signal pixel_x_std: std_logic_vector(9 downto 0);
   signal pixel_y_std: std_logic_vector(9 downto 0);

   signal LM_ULM_start, LM_ULM_ready: std_logic;
   signal LM_ULM_stopped, LM_ULM_continue: std_logic;
   signal LM_ULM_done: std_logic;
   signal LM_ULM_base_address: std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
   signal LM_ULM_upper_limit: std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
   signal LM_ULM_load_unload: std_logic;

   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 --
================================================================================
==================
89    begin
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
```

```
   text: entity work.font_test_gen(arch)
      port map (clk=>Clk,
                video_on=>hdmi_enable_out,
                pixel_x=>pixel_x_std,
                pixel_y=>pixel_y_std,
                rgb_text_out=>rgb_text
                );

      --
================================================================================
================
   -- turn on the appropriate color depending on the current pixel position.
      process (hdmi_enable_out, wall1_on, wall2_on, wall3_on, wall4_on, bar_on,
asteroid2_on, asteroid1_on, wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, bar_rgb,
asteroid2_rgb, asteroid1_rgb)
      begin
      if (hdmi_enable_out = '0') then
         graph_rgb <= "110"; -- blank
      else
         if (wall1_on = '1') then
            graph_rgb <= wall1_rgb;
         elsif (wall2_on = '1') then
            graph_rgb <= wall2_rgb;
         elsif (wall3_on = '1') then
            graph_rgb <= wall3_rgb;
         elsif (wall4_on = '1') then
            graph_rgb <= wall4_rgb;
         elsif (bar_on = '1') then
            graph_rgb <= bar_rgb;
         elsif (asteroid2_on = '1') then
            graph_rgb <= asteroid2_rgb;
         elsif (asteroid1_on = '1') then
            graph_rgb <= asteroid1_rgb;
         elsif (missile_on = '1') then
            graph_rgb <= missile_rgb;
         -- elsif (missile_on = '1') then
         --    graph_rgb <= rgb_text;
         else
            graph_rgb <= "000"; -- Black bkgnd
            graph_rgb <= rgb_text;
         end if;
      end if;
   end process;


--    hdmi_red <= std_logic_vector(resize(pixel_x, 8)) when sw_r = '1' else (others
=> '0');
--    hdmi_green <= std_logic_vector(resize(pixel_y, 8)) when sw_g = '1' else (others
=> '0');
   hdmi_red <= "11111111" when graph_rgb(0) = '1' else (others => '0');
   hdmi_green <= "11111111" when graph_rgb(1) = '1' else (others => '0');
   hdmi_blue <= "11111111" when graph_rgb(2) = '1' else (others => '0');

   hdmi_enable <= hdmi_enable_out;

end beh;
```

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

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

entity asteroid is
   generic(
      asteroid_velocity: integer := 5
   );

   port(
      clk, reset: in std_logic;
      -- btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
      video_on: in std_logic;
      pixel_x, pixel_y: in std_logic_vector(9 downto 0);
      asteroid_on_out: out std_logic;
      asteroid_rgb_out: out std_logic_vector(2 downto 0)
   );
end asteroid;

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

architecture rtl of asteroid is

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

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

-- Screen dimensions
   constant MAX_X: integer := 640;
   constant MAX_Y: integer := 480;

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

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

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

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

-- Paddle left, right, top, bottom and height -- left & right are constant. Top &
bottom are signals to allow movement. bar_y_t driven by register below.
   --constant BAR_X_L: integer := 600;--R2
   --constant BAR_X_R: integer := 603;--R2
```

```
   -- signal bar_x_L,bar_x_R: unsigned(9 downto 0); --R2
   -- signal bar_y_t, bar_y_b: unsigned(9 downto 0);
   -- constant BAR_Y_SIZE: integer := 72;
   -- constant BAR_X_SIZE: integer := 8; --R2

-- -- Reg to track top boundary (x position is fixed)
--    signal bar_y_reg, bar_y_next: unsigned( 9 downto 0);

-- -- Reg to track right boundary --R2
--    signal bar_x_reg, bar_x_next: unsigned( 9 downto 0); --R2

-- Bar moving velocity when a button is pressed -- the amount the bar is moved.
   -- constant BAR_V: integer:= 4;

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

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

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

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

-- round ball image
   type rom_type is array(0 to 15) of std_logic_vector(0 to 15); ----------Changed
from array(0 to 7)
   constant BALL_ROM: rom_type:= (
      "0001111111110000",
      "0010110011011000",
      "0111101101110100",
      "1111011110111110",
      "1011011011011111",
      "1111101111011011",
      "1110110110111111",
      "1101111001111111",
      "1101111111011111",
      "1111011111111111",
      "1011101111111111",
      "1110110011111111",
      "0111111111111110",
      "0010111101101100",
      "0001101111111000",
      "0000111111110000");

      -- Testing
      -- 0 is to paint the background and 1 will be to use the ball
      -- Can make more than one bit to make characters and other sprites.


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

-- object output signals -- new signal to indicate if scan coord is within ball
   -- signal wall1_on, wall2_on, wall3_on, wall4_on, bar_on, sq_ball_on, rd_ball_on:
std_logic;
   -- signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, bar_rgb, ball_rgb:
std_logic_vector(2 downto 0);
   signal wall1_on, wall2_on, wall3_on, wall4_on, sq_ball_on, rd_ball_on: std_logic;
   signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, ball_rgb: std_logic_vector(2
downto 0);

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

```
================
   begin

   asteroid_on_out <= rd_ball_on;
   asteroid_rgb_out <= ball_rgb;

   process (clk, reset)
      begin
      if (reset = '1') then
         -- bar_y_reg <= (others => '0');
         -- bar_x_reg <= (others => '0');--R2
         ball_x_reg <= (others => '0');
         ball_y_reg <= (others => '0');
         x_delta_reg <= ("0000000100");
         y_delta_reg <= ("0000000100");
      elsif (clk'event and clk = '1') then
         -- bar_y_reg <= bar_y_next;
         -- bar_x_reg <= bar_x_next;--R2
         ball_x_reg <= ball_x_next;
         ball_y_reg <= ball_y_next;
         x_delta_reg <= x_delta_next;
         y_delta_reg <= y_delta_next;
      end if;
   end process;

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

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

--
================================================================================
================
-- -- wall1 left vertical stripe
--    wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
convert pix_x to pix_y to make horizontal
--    wall1_rgb <= "011"; -- paddle colors blue
-- -- wall2 right vertical stripe
--    wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
--    wall2_rgb <= "011"; -- paddle colors blue
-- -- wall3 left vertical stripe
--    wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
convert pix_x to pix_y to make horizontal
--    wall3_rgb <= "011"; -- paddle colors blue
-- -- wall4 right vertical stripe
--    wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
--    wall4_rgb <= "011"; -- paddle colors blue

--
================================================================================
================
-- -- pixel within paddle
--    bar_x_L <= bar_x_reg;--R2
--    bar_x_R <= bar_x_L + BAR_X_SIZE - 1;--R2

--    bar_y_t <= bar_y_reg;
--    bar_y_b <= bar_y_t + BAR_Y_SIZE - 1;
--    bar_on <= '1' when (BAR_X_L <= pix_x) and (pix_x <= BAR_X_R) and (bar_y_t <=
pix_y) and (pix_y <= bar_y_b) else '0';
--    bar_rgb <= "100"; -- Red color

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

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

```

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

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

-- Get row data
   rom_data <= BALL_ROM(to_integer(rom_addr));

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

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

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

-- Set the value of the next ball position according to the boundaries.
   -- process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_t, ball_y_b, bar_y_t, bar_y_b)
   process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_t, ball_y_b)
      begin
      x_delta_next <= x_delta_reg;
      y_delta_next <= y_delta_reg;

-- Ball reached top, make offset positive
      if ( ball_y_t < WALL4_X_B ) then --MAKE WALL4
         y_delta_next <= BALL_V_P;

-- Reached bottom, make negative
      elsif (ball_y_b > (WALL3_X_T - 1)) then --MAKE WALL3
         y_delta_next <= BALL_V_N;

-- Reach wall1, bounce back
      elsif (ball_x_l <= WALL1_X_R ) then
         x_delta_next <= BALL_V_P;
-- Reach wall2, bounce back
      elsif (ball_x_r > WALL2_X_L) then
         x_delta_next <= BALL_V_N;

-- -- Right corner of ball inside bar
--       elsif ((BAR_X_L <= ball_x_r) and (ball_x_r <= BAR_X_R)) then

-- -- Some portion of ball hitting paddle, reverse direction
--          if ((bar_y_t <= ball_y_b) and (ball_y_t <= bar_y_b)) then
--             x_delta_next <= BALL_V_N;
--          end if;
      end if;
   end process;

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

end rtl;


```

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

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

entity asteroid is
   generic(
      asteroid_velocity: integer := 5
   );

   port(
      clk, reset: in std_logic;
      -- btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
      video_on: in std_logic;
      pixel_x, pixel_y: in std_logic_vector(9 downto 0);
      asteroid_on_out: out std_logic;
      asteroid_rgb_out: out std_logic_vector(2 downto 0)
   );
end asteroid;

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

architecture rtl of asteroid is

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

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

-- Screen dimensions
   constant MAX_X: integer := 640;
   constant MAX_Y: integer := 480;

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

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

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

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

-- Paddle left, right, top, bottom and height -- left & right are constant. Top &
bottom are signals to allow movement. bar_y_t driven by register below.
   --constant BAR_X_L: integer := 600;--R2
   --constant BAR_X_R: integer := 603;--R2
```

```
   -- signal bar_x_L,bar_x_R: unsigned(9 downto 0); --R2
   -- signal bar_y_t, bar_y_b: unsigned(9 downto 0);
   -- constant BAR_Y_SIZE: integer := 72;
   -- constant BAR_X_SIZE: integer := 8; --R2

-- -- Reg to track top boundary (x position is fixed)
--    signal bar_y_reg, bar_y_next: unsigned( 9 downto 0);

-- -- Reg to track right boundary --R2
--    signal bar_x_reg, bar_x_next: unsigned( 9 downto 0); --R2

-- Bar moving velocity when a button is pressed -- the amount the bar is moved.
   -- constant BAR_V: integer:= 4;

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

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

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

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

-- round ball image
   type rom_type is array(0 to 15) of std_logic_vector(0 to 15); ----------Changed
from array(0 to 7)
   constant BALL_ROM: rom_type:= (
      "0001111111110000",
      "0010110011011000",
      "0111101101110100",
      "1111011110111110",
      "1011011011011111",
      "1111101111011011",
      "1110110110111111",
      "1101111001111111",
      "1101111111011111",
      "1111011111111111",
      "1011101111111111",
      "1110110011111111",
      "0111111111111110",
      "0010111101101100",
      "0001101111111000",
      "0000111111110000");

      -- Testing
      -- 0 is to paint the background and 1 will be to use the ball
      -- Can make more than one bit to make characters and other sprites.


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

-- object output signals -- new signal to indicate if scan coord is within ball
   -- signal wall1_on, wall2_on, wall3_on, wall4_on, bar_on, sq_ball_on, rd_ball_on:
std_logic;
   -- signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, bar_rgb, ball_rgb:
std_logic_vector(2 downto 0);
   signal wall1_on, wall2_on, wall3_on, wall4_on, sq_ball_on, rd_ball_on: std_logic;
   signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb, ball_rgb: std_logic_vector(2
downto 0);

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

```
================
   begin

   asteroid_on_out <= rd_ball_on;
   asteroid_rgb_out <= ball_rgb;

   process (clk, reset)
      begin
      if (reset = '1') then
         -- bar_y_reg <= (others => '0');
         -- bar_x_reg <= (others => '0');--R2
         ball_x_reg <= (others => '0');
         ball_y_reg <= (others => '0');
         x_delta_reg <= ("0000000100");
         y_delta_reg <= ("0000000100");
      elsif (clk'event and clk = '1') then
         -- bar_y_reg <= bar_y_next;
         -- bar_x_reg <= bar_x_next;--R2
         ball_x_reg <= ball_x_next;
         ball_y_reg <= ball_y_next;
         x_delta_reg <= x_delta_next;
         y_delta_reg <= y_delta_next;
      end if;
   end process;

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

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

--
================================================================================
================
-- -- wall1 left vertical stripe
--    wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
convert pix_x to pix_y to make horizontal
--    wall1_rgb <= "011"; -- paddle colors blue
-- -- wall2 right vertical stripe
--    wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
--    wall2_rgb <= "011"; -- paddle colors blue
-- -- wall3 left vertical stripe
--    wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
convert pix_x to pix_y to make horizontal
--    wall3_rgb <= "011"; -- paddle colors blue
-- -- wall4 right vertical stripe
--    wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
--    wall4_rgb <= "011"; -- paddle colors blue

--
================================================================================
================
-- -- pixel within paddle
--    bar_x_L <= bar_x_reg;--R2
--    bar_x_R <= bar_x_L + BAR_X_SIZE - 1;--R2

--    bar_y_t <= bar_y_reg;
--    bar_y_b <= bar_y_t + BAR_Y_SIZE - 1;
--    bar_on <= '1' when (BAR_X_L <= pix_x) and (pix_x <= BAR_X_R) and (bar_y_t <=
pix_y) and (pix_y <= bar_y_b) else '0';
--    bar_rgb <= "100"; -- Red color

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

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

```

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

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

-- Get row data
   rom_data <= BALL_ROM(to_integer(rom_addr));

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

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

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

-- Set the value of the next ball position according to the boundaries.
   -- process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_t, ball_y_b, bar_y_t, bar_y_b)
   process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r, ball_y_t, ball_y_b)
      begin
      x_delta_next <= x_delta_reg;
      y_delta_next <= y_delta_reg;

-- Ball reached top, make offset positive
      if ( ball_y_t < WALL4_X_B ) then --MAKE WALL4
         y_delta_next <= BALL_V_P;

-- Reached bottom, make negative
      elsif (ball_y_b > (WALL3_X_T - 1)) then --MAKE WALL3
         y_delta_next <= BALL_V_N;

-- Reach wall1, bounce back
      elsif (ball_x_l <= WALL1_X_R ) then
         x_delta_next <= BALL_V_P;
-- Reach wall2, bounce back
      elsif (ball_x_r > WALL2_X_L) then
         x_delta_next <= BALL_V_N;

-- -- Right corner of ball inside bar
--       elsif ((BAR_X_L <= ball_x_r) and (ball_x_r <= BAR_X_R)) then

-- -- Some portion of ball hitting paddle, reverse direction
--          if ((bar_y_t <= ball_y_b) and (ball_y_t <= bar_y_b)) then
--             x_delta_next <= BALL_V_N;
--          end if;
      end if;
   end process;

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

end rtl;


```

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

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

entity walls is
   port(
      clk, reset: in std_logic;
      -- btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
      video_on: in std_logic;
      pixel_x, pixel_y: in std_logic_vector(9 downto 0);
      wall1_on_out, wall2_on_out, wall3_on_out, wall4_on_out: out std_logic;
      wall1_rgb_out, wall2_rgb_out, wall3_rgb_out, wall4_rgb_out: out
std_logic_vector(2 downto 0);
      Left_Wall_Out, Right_Wall_Out, Bottom_Wall_Out, Top_Wall_Out: out
std_logic_vector(9 downto 0)
   );
end walls;

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

architecture rtl of walls is

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

-- x, y coordinates (0,0 to (639, 479)
   signal pix_x, pix_y: unsigned(9 downto 0);
   signal wall1_on, wall2_on, wall3_on, wall4_on: std_logic;
   signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb: std_logic_vector(2 downto 0);
   signal Left_Wall, Right_Wall, Bottom_Wall, Top_Wall: std_logic_vector(9 downto 0);

-- Screen dimensions
   constant MAX_X: integer := 640;
   constant MAX_Y: integer := 480;

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

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

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

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

--
================================================================================
================
   begin
   wall1_on_out <= wall1_on;
```

```
   wall2_on_out <= wall2_on;
   wall3_on_out <= wall3_on;
   wall4_on_out <= wall4_on;
   -- bar_on_out <= rom_bar_on;
   -- asteroid2_on_out <= rd_ball_on;

   wall1_rgb_out <= wall1_rgb;
   wall2_rgb_out <= wall2_rgb;
   wall3_rgb_out <= wall3_rgb;
   wall4_rgb_out <= wall4_rgb;
   Left_Wall_Out <= Left_Wall;
   Right_Wall_Out <= Right_Wall;
   Bottom_Wall_Out <= Bottom_Wall;
   Top_Wall_Out <= Top_Wall;
   Left_Wall <= std_logic_vector(to_unsigned(WALL1_X_R,10));
   Right_Wall <= std_logic_vector(to_unsigned(WALL2_X_L,10));
   Bottom_Wall <= std_logic_vector(to_unsigned(WALL3_X_T,10));
   Top_Wall <= std_logic_vector(to_unsigned(WALL4_X_B,10));

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

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

--
================================================================================
================
-- wall1 left vertical stripe
   wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
convert pix_x to pix_y to make horizontal
   wall1_rgb <= "011"; -- paddle colors blue
-- wall2 right vertical stripe
   wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
   wall2_rgb <= "011"; -- paddle colors blue
-- wall3 left vertical stripe
   wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
convert pix_x to pix_y to make horizontal
   wall3_rgb <= "011"; -- paddle colors blue
-- wall4 right vertical stripe
   wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
   wall4_rgb <= "011"; -- paddle colors blue

end rtl;
```

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

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

entity walls is
   port(
      clk, reset: in std_logic;
      -- btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
      video_on: in std_logic;
      pixel_x, pixel_y: in std_logic_vector(9 downto 0);
      wall1_on_out, wall2_on_out, wall3_on_out, wall4_on_out: out std_logic;
      wall1_rgb_out, wall2_rgb_out, wall3_rgb_out, wall4_rgb_out: out
std_logic_vector(2 downto 0);
      Left_Wall_Out, Right_Wall_Out, Bottom_Wall_Out, Top_Wall_Out: out
std_logic_vector(9 downto 0)
   );
end walls;

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

architecture rtl of walls is

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

-- x, y coordinates (0,0 to (639, 479)
   signal pix_x, pix_y: unsigned(9 downto 0);
   signal wall1_on, wall2_on, wall3_on, wall4_on: std_logic;
   signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb: std_logic_vector(2 downto 0);
   signal Left_Wall, Right_Wall, Bottom_Wall, Top_Wall: std_logic_vector(9 downto 0);

-- Screen dimensions
   constant MAX_X: integer := 640;
   constant MAX_Y: integer := 480;

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

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

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

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

--
================================================================================
================
   begin
   wall1_on_out <= wall1_on;
```

```
   wall2_on_out <= wall2_on;
   wall3_on_out <= wall3_on;
   wall4_on_out <= wall4_on;
   -- bar_on_out <= rom_bar_on;
   -- asteroid2_on_out <= rd_ball_on;

   wall1_rgb_out <= wall1_rgb;
   wall2_rgb_out <= wall2_rgb;
   wall3_rgb_out <= wall3_rgb;
   wall4_rgb_out <= wall4_rgb;
   Left_Wall_Out <= Left_Wall;
   Right_Wall_Out <= Right_Wall;
   Bottom_Wall_Out <= Bottom_Wall;
   Top_Wall_Out <= Top_Wall;
   Left_Wall <= std_logic_vector(to_unsigned(WALL1_X_R,10));
   Right_Wall <= std_logic_vector(to_unsigned(WALL2_X_L,10));
   Bottom_Wall <= std_logic_vector(to_unsigned(WALL3_X_T,10));
   Top_Wall <= std_logic_vector(to_unsigned(WALL4_X_B,10));

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

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

--
================================================================================
================
-- wall1 left vertical stripe
   wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
convert pix_x to pix_y to make horizontal
   wall1_rgb <= "011"; -- paddle colors blue
-- wall2 right vertical stripe
   wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
   wall2_rgb <= "011"; -- paddle colors blue
-- wall3 left vertical stripe
   wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
convert pix_x to pix_y to make horizontal
   wall3_rgb <= "011"; -- paddle colors blue
-- wall4 right vertical stripe
   wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
   wall4_rgb <= "011"; -- paddle colors blue

end rtl;
```

```
----------------------------------------------------------------------------------
-- Company: University of New Mexico
-- Engineer: Professor Jim Plusquellic, Copyright Univ. of New Mexico
--
-- Create Date:
-- Design Name:
-- Module Name:    LoadUnLoadMem - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------

-- LoadUnLoadMem securely transfers data into or out of PNL_BRAM using GPIO registers

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

library work;
use work.DataTypes_pkg.all;

entity LoadUnLoadMem is
   port(
      Clk: in  std_logic;
      RESET: in std_logic;
      start: in std_logic;
      ready: out std_logic;
      load_unload: in std_logic;
      stopped: out std_logic;
      continue: in std_logic;
      done: in std_logic;
      base_address: in std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
      upper_limit: in std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
      CP_in_word: in std_logic_vector(WORD_SIZE_NB-1 downto 0);
      CP_out_word: out std_logic_vector(WORD_SIZE_NB-1 downto 0);
      PNL_BRAM_addr: out std_logic_vector(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);
      PNL_BRAM_din: out std_logic_vector(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
      PNL_BRAM_dout: in std_logic_vector(PNL_BRAM_DBITS_WIDTH_NB-1 downto 0);
      PNL_BRAM_we: out std_logic_vector(0 to 0)
      );
end LoadUnLoadMem;

architecture beh of LoadUnLoadMem is
   type state_type is (idle, load_mem, unload_mem, wait_load_unload, wait_done);
   signal state_reg, state_next: state_type;

   signal ready_reg, ready_next: std_logic;

   signal PNL_BRAM_addr_reg, PNL_BRAM_addr_next: unsigned(PNL_BRAM_ADDR_SIZE_NB-1
downto 0);
   signal PNL_BRAM_upper_limit_reg, PNL_BRAM_upper_limit_next:
unsigned(PNL_BRAM_ADDR_SIZE_NB-1 downto 0);

   begin

--
================================================================================
========
-- State and register logic
--
================================================================================
========
   process(Clk, RESET)
      begin
      if ( RESET = '1' ) then
         state_reg <= idle;
         ready_reg <= '1';
         PNL_BRAM_addr_reg <= (others=>'0');
         PNL_BRAM_upper_limit_reg <= (others=>'0');
      elsif ( Clk'event and Clk = '1' ) then
```

```
         state_reg <= state_next;
         ready_reg <= ready_next;
         PNL_BRAM_addr_reg <= PNL_BRAM_addr_next;
         PNL_BRAM_upper_limit_reg <= PNL_BRAM_upper_limit_next;
      end if;
   end process;

--
================================================================================
========
-- Combo logic
--
================================================================================
========
   process (state_reg, start, ready_reg, load_unload, PNL_BRAM_addr_reg,
PNL_BRAM_upper_limit_reg,
      PNL_BRAM_dout, CP_in_word, continue, base_address, upper_limit, done)
      begin
      state_next <= state_reg;
      ready_next <= ready_reg;

      PNL_BRAM_addr_next <= PNL_BRAM_addr_reg;
      PNL_BRAM_upper_limit_next <= PNL_BRAM_upper_limit_reg;

      PNL_BRAM_we <= "0";
      PNL_BRAM_din <= (others=>'0');
      CP_out_word <= (others=>'0');

      stopped <= '0';

      case state_reg is

-- ====================
         when idle =>
            ready_next <= '1';

            if ( start = '1' ) then
               ready_next <= '0';

-- Latch the 'base_address' and 'upper_limit' at the instant 'start' is asserted.
NOTE: These signals MAY BE SET
-- BACK TO all 0's after the 'start' signal is received.
               PNL_BRAM_addr_next <= unsigned(base_address);
               PNL_BRAM_upper_limit_next <= unsigned(upper_limit);

               if ( load_unload = '0' ) then
                  state_next <= load_mem;
               else
                  state_next <= unload_mem;
               end if;
            end if;

-- ====================
-- Write value to memory location
         when load_mem =>

-- Signal C program that we are ready to receive a word. Once ready ('continue'
becomes '1'), transfer and complete handshake.
            stopped <= '1';
            if ( done = '0' ) then
               if ( continue = '1' ) then
                  PNL_BRAM_we <= "1";
                  PNL_BRAM_din <= (PNL_BRAM_DBITS_WIDTH_NB-1 downto WORD_SIZE_NB =>
'0') & CP_in_word;

-- Wait handshake signals
                  state_next <= wait_load_unload;
               end if;

-- Handle case where C program has nothing to store.
            else
               state_next <= wait_done;
            end if;

-- ====================
-- Get value at memory location
         when unload_mem =>

```

```
-- Put the PNL BRAM word on CP_out_word. Do NOT do this by default for security reasons.
            CP_out_word <= PNL_BRAM_dout(WORD_SIZE_NB-1 downto 0);

-- Signal C program that we are ready to deliver a word. Once it reads the word, it sets 'continue' to '1'.
            stopped <= '1';
            if ( continue = '1' ) then

-- Wait handshake signals
               state_next <= wait_load_unload;
            end if;

-- Handle case where C program does not want to read any data.
            if ( done = '1' ) then
               state_next <= wait_done;
            end if;

-- =====================
-- Complete handshake and update addresses
         when wait_load_unload =>

-- C program holds 'continue' at 1 until it sees 'stopped' go to 0, and then it writes a '0' to continue. It also writes
-- 'done' with a '1' when last transfer is made.
            if ( continue = '0' ) then

-- Done collecting C program transmitted words. Force a finish if the upper limit has been reached. This will protect the memory
-- from overruns (reading or writing).
               if ( done = '1' ) then
                  state_next <= wait_done;
               elsif ( PNL_BRAM_addr_reg = PNL_BRAM_upper_limit_reg ) then
                  state_next <= idle;
               else
                  PNL_BRAM_addr_next <= PNL_BRAM_addr_reg + 1;
                  if ( load_unload = '0' ) then
                     state_next <= load_mem;
                  else
                     state_next <= unload_mem;
                  end if;
               end if;
            end if;

-- =====================
-- Wait for 'done' to return to 0 before returning to idle, if it was set by the C program to exit early.
         when wait_done =>
            if ( done = '0' ) then
               state_next <= idle;
            end if;

      end case;
   end process;

-- Use 'look-ahead' signal for BRAM address.
   PNL_BRAM_addr <= std_logic_vector(PNL_BRAM_addr_next);
   ready <= ready_reg;

end beh;
```

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

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

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

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

   );
end missile;

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

architecture rtl of missile is

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

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

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

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

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

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

```

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

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

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

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

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

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


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

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


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

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


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

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

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

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

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

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

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

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

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

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

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

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


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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   end process;

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


end rtl;



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

```
----------------------------------------------------------------------------------
-- Engineer: Zachary Montoya
-- Submitted Date: 12-15-22
-- Module Name:    Top - Behavioral
-- Project Name:   Asteroids
-- Target Devices: Zybo Z7-10
-- Tool versions: Vivado 2020.2
--
-- Comment:
--         This file was entirely created the engineer listed above.
--         EXCEPT FOR PARTS OF LINES 236 and 241 SPECIFICALLY THE COMPARISON
ORIENTATION
--         THESE PARTS WERE CREATED BY THE FOLLOWING
--         Engineer: Anindya Bal
--
----------------------------------------------------------------------------------

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

entity space_ship is
   port(
      clk, reset: in std_logic;
      btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
      sw: in std_logic_vector(0 downto 0);
      video_on: in std_logic;
      pixel_x, pixel_y: in std_logic_vector(9 downto 0);
      bar_on_out: out std_logic;
      bar_rgb_out: out std_logic_vector(2 downto 0);
      bar_x_reg_out, bar_y_reg_out: out unsigned( 9 downto 0);
      rom_selector_out: out std_logic_vector(2 downto 0);
      graph_rgb: out std_logic_vector(2 downto 0);
      fire_out,fire_btn_signal_out: out std_logic;
      fire_ready_out: in std_logic

   );
end space_ship;

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

architecture rtl of space_ship is

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

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

-- Screen dimensions
   constant MAX_X: integer := 640;
   constant MAX_Y: integer := 480;

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

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

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

-- WALL4 - TOP
```

```
   constant WALL4_X_T: integer := 0;
   constant WALL4_X_B: integer := 20;
-- Bar moving velocity when a button is pressed -- the amount the bar is moved.

--
================================================================================
================
-- BAR STUFF
   -- Paddle left, right, top, bottom and height -- left & right are constant. Top &
bottom are signals to allow movement. bar_y_t driven by register below.
   signal bar_x_L,bar_x_R: unsigned(9 downto 0); --R2
   signal bar_y_t, bar_y_b: unsigned(9 downto 0);
   constant BAR_SIZE: integer := 16; --R2
   -- Reg to track top boundary (x position is fixed)
   -- signal bar_y_reg: unsigned( 9 downto 0) := "1111001110";
   -- signal bar_x_reg: unsigned( 9 downto 0) := "1111001110";
   -- signal bar_x_next, bar_y_next: unsigned( 9 downto 0);
   signal bar_x_reg, bar_y_reg, bar_x_next, bar_y_next: unsigned( 9 downto 0);
   -- signal bar_x_reg_out, bar_y_reg_out: unsigned( 9 downto 0);

-- ball movement can be pos or neg
   constant BAR_V: integer:= 3;

-- round TOP image
   type spaceship_rom_type is array(0 to 15) of std_logic_vector(0 to 15); ----------
Changed from array(0 to 7)
   constant SPACESHIP_UP_ROM: spaceship_rom_type:= (
         "0000000110000000",
         "0000000110000000",
         "0000000110000000",
         "0000001111000000",
         "0000001111000000",
         "1100001111000011",
         "1100001111000011",
         "1100011001100011",
         "1100011001100011",
         "1111111111111111",
         "1111111111111111",
         "1100011001100011",
         "1100011001100011",
         "1100001001000011",
         "0001110110111000",
         "0001011001101000");

   -- round BOTTOM image
   constant SPACESHIP_DOWN_ROM: spaceship_rom_type:= (
         "0001011001101000",
         "0001110110111000",
         "1100001001000011",
         "1100011001100011",
         "1100011001100011",
         "1111111111111111",
         "1111111111111111",
         "1100011001100011",
         "1100011001100011",
         "1100001111000011",
         "1100001111000011",
         "0000001111000000",
         "0000001111000000",
         "0000000110000000",
         "0000000110000000",
         "0000000110000000");

   constant SPACESHIP_RIGHT_ROM: spaceship_rom_type:= (
         "0011111111100000",
         "0011111111100000",
         "0000011000000000",
         "1100011000000000",
         "0100011000000000",
         "1101111110000000",
         "1011111111111000",
         "0100011001111111",
         "0100011001111111",
         "1011111111111000",
         "1101111110000000",
         "0100011000000000",
         "1100011000000000",
         "0000011000000000",
         "0011111111100000",
```

```
         "0011111111100000");

         constant SPACESHIP_LEFT_ROM: spaceship_rom_type:= (
         "0000011111111100",
         "0000011111111100",
         "0000000001100000",
         "0000000001100011",
         "0000000001100010",
         "0000000001111011",
         "0001111111111101",
         "1111111001100010",
         "1111111001100010",
         "0001111111111101",
         "0000000001111011",
         "0000000001100010",
         "0000000001100011",
         "0000000001100000",
         "0000011111111100",
         "0000011111111100");

   signal spaceship_rom_addr, spaceship_rom_col: unsigned(3 downto 0); ----------
----------Changed to 4 bits from unsigned (2 downto 0)
   signal spaceship_rom_data: std_logic_vector(15 downto 0); ----------------------
-Changed from (7 downto 0)
   signal spaceship_rom_bit: std_logic;
-- --
================================================================================
=================

   signal bar_on, rom_bar_on: std_logic;
   signal bar_rgb: std_logic_vector(2 downto 0);
   signal rom_selector: std_logic_vector(2 downto 0);
   signal fire_next, fire_reg: std_logic;
   signal fire_btn_signal_reg: std_logic := '0';
   signal fire_btn_signal_next: std_logic;
   -- signal rom_selector_out:std_logic_vector(1 downto 0);
--
================================================================================
=================
   begin
   bar_on_out <= rom_bar_on;
   bar_rgb_out <= bar_rgb;
   bar_x_reg_out <= bar_x_reg;
   bar_y_reg_out <= bar_y_reg;
   rom_selector_out <= rom_selector;
   fire_out <= fire_next;
   fire_btn_signal_out <= fire_btn_signal_reg;

   process (clk, reset)
      begin
      if (reset = '1') then
         -- bar_y_reg <= (others => '0');
         -- bar_x_reg <= (others => '0');--R2
         bar_y_reg <= (to_unsigned(240,10));
         bar_x_reg <= (to_unsigned(320,10));--R2
         fire_reg <= '0';
         fire_btn_signal_reg <= '0';
      elsif (clk'event and clk = '1') then
         bar_y_reg <= bar_y_next;
         bar_x_reg <= bar_x_next;--R2
         fire_reg <= fire_next;
         fire_btn_signal_reg <= fire_btn_signal_next;
      end if;
   end process;
--
================================================================================
=================
   --FIRE CONTROL
   process (btn, fire_reg , fire_ready_out, fire_btn_signal_reg)
   begin
      fire_next <= fire_reg;
      fire_btn_signal_next <= fire_btn_signal_reg;
      if (btn(0) = '1' and fire_btn_signal_reg = '0') then
      -- if (btn(0) = '1' ) then
         fire_next <= '1';
         fire_btn_signal_next <= '1';
      elsif (fire_ready_out = '1') then
         fire_btn_signal_next <= '0';
         fire_next <= '0';
```

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

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

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

entity walls is
   port(
      clk, reset: in std_logic;
      -- btn: in std_logic_vector(3 downto 0); --R2 increasing the BTN array to 3
      video_on: in std_logic;
      pixel_x, pixel_y: in std_logic_vector(9 downto 0);
      wall1_on_out, wall2_on_out, wall3_on_out, wall4_on_out: out std_logic;
      wall1_rgb_out, wall2_rgb_out, wall3_rgb_out, wall4_rgb_out: out
std_logic_vector(2 downto 0);
      Left_Wall_Out, Right_Wall_Out, Bottom_Wall_Out, Top_Wall_Out: out
std_logic_vector(9 downto 0)
   );
end walls;

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

architecture rtl of walls is

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

-- x, y coordinates (0,0 to (639, 479)
   signal pix_x, pix_y: unsigned(9 downto 0);
   signal wall1_on, wall2_on, wall3_on, wall4_on: std_logic;
   signal wall1_rgb, wall2_rgb, wall3_rgb, wall4_rgb: std_logic_vector(2 downto 0);
   signal Left_Wall, Right_Wall, Bottom_Wall, Top_Wall: std_logic_vector(9 downto 0);

-- Screen dimensions
   constant MAX_X: integer := 640;
   constant MAX_Y: integer := 480;

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

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

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

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

--
================================================================================
================
   begin
   wall1_on_out <= wall1_on;
```

```
   wall2_on_out <= wall2_on;
   wall3_on_out <= wall3_on;
   wall4_on_out <= wall4_on;
   -- bar_on_out <= rom_bar_on;
   -- asteroid2_on_out <= rd_ball_on;

   wall1_rgb_out <= wall1_rgb;
   wall2_rgb_out <= wall2_rgb;
   wall3_rgb_out <= wall3_rgb;
   wall4_rgb_out <= wall4_rgb;
   Left_Wall_Out <= Left_Wall;
   Right_Wall_Out <= Right_Wall;
   Bottom_Wall_Out <= Bottom_Wall;
   Top_Wall_Out <= Top_Wall;
   Left_Wall <= std_logic_vector(to_unsigned(WALL1_X_R,10));
   Right_Wall <= std_logic_vector(to_unsigned(WALL2_X_L,10));
   Bottom_Wall <= std_logic_vector(to_unsigned(WALL3_X_T,10));
   Top_Wall <= std_logic_vector(to_unsigned(WALL4_X_B,10));

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

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

--
================================================================================
================
-- wall1 left vertical stripe
   wall1_on <= '1' when (WALL1_X_L <= pix_x) and (pix_x <= WALL1_X_R) else '0'; --
convert pix_x to pix_y to make horizontal
   wall1_rgb <= "011"; -- paddle colors blue
-- wall2 right vertical stripe
   wall2_on <= '1' when (WALL2_X_L <= pix_x) and (pix_x <= WALL2_X_R) else '0';
   wall2_rgb <= "011"; -- paddle colors blue
-- wall3 left vertical stripe
   wall3_on <= '1' when (WALL3_X_T <= pix_y) and (pix_y <= WALL3_X_B) else '0'; --
convert pix_x to pix_y to make horizontal
   wall3_rgb <= "011"; -- paddle colors blue
-- wall4 right vertical stripe
   wall4_on <= '1' when (WALL4_X_T <= pix_y) and (pix_y <= WALL4_X_B) else '0';
   wall4_rgb <= "011"; -- paddle colors blue

end rtl;
```